Type Theory and Decision Procedures

Pierre-Yves Strub

École Polytechnique

Thèse

présentée pour obtenir

Le Grade de Docteur en Informatique
de l'École Polytechnique

par

Pierre-Yves Strub

Théorie des Types et Procédures de Décision

Soutenue le 2 juillet 2008
devant la commission d'examen composée de:

Gérard Huet	Président du Jury
Gilles Barthe	Rapporteur
José Meseguer	
Natarajan Shankar	
Frédéric Blanqui	Examinateur
Christine Paulin-Mohring	
Benjamin Werner	
Jean-Pierre Jouannaud	Directeur

Abstract

The goal of this thesis is to develop a logical system in which formal proofs of mathematical statements can be carried out in a way which is close to mathematical practice.

Our main contribution is the definition and study of a new calculus, the Calculus of Congruent and Inductive Constructions, an extension of the Calculus of Inductive Constructions (CIC) integrating in its computational part the entailment relation - via decision procedures - of a first order theory over equality. A major technical innovation of this work lies in the computational mechanism: goals are sent to the decision procedure together with a set of user hypotheses available from the current context.

Our main results show that this extension of CIC does not compromise its main properties: confluence, strong normalization, consistency and decidability of proof checking are all preserved (as soon as the incorporated theory is itself decidable). As such, our calculus can be seen as a decidable restriction of the Extensional Calculus of Constructions. It can therefore serve as the basis for an extension of the Coq proof assistant.

Résumé

Le but de cette thèse est l'étude d'un système logique formel dans lequel les preuves formelles de propriétés mathématiques sont menées dans un style plus proches des pratiques des mathématiciens.

Notre principal apport est la définition et l'étude du Calcul des Constructions Inductives Congruentes, une extension du Calcul des Constructions Inductives (CIC), intégrant au sein de son mécanisme de calcul des procédures de décisions pour des théories equationnelles au premier ordre.

Nous montrons que ce calcul possède toutes les propriétés attendues : confluence, normalisation forte, cohérence logique et décidabilité de la vérification de types sont préservées. En tant que tel, notre calcul peut être vu comme une restriction décidable du Calcul des Constructions Extentionnelles et peut servir comme base pour l'extension de l'assistant à la preuve Coq.

This thesis has been partly supported by *Fondation d'Entreprise EADS*.

This thesis has been partly supported by *France Telecom*.

Contents

List of Figures

CHAPTER 1

Introduction

> Ce n'est pas une démonstration proprement dite, [...] c'est une vérification. [...] La vérification diffère précisément de la véritable démonstration, parce qu'elle est purement analytique et parce qu'elle est stérile.
>
> Henri Poincaré, on the proof of $2 + 2 = 4$
> *La science et l'Hypothèse*, 1902

The goal of this thesis is to develop a logical system in which formal proofs of mathematical statements can be carried out in a way which is close to mathematical practice.

What is a formal proof? Up the end of the XIX$^{\text{th}}$ century, no formal description of reasoning existed: mathematical proofs were made by using an intuitive notion of proof or validity. Frege and Peano initiated a new discipline: *formal mathematics* or *formal logic*. Formal mathematics is the description of *formal rules* - permitting the construction of complex mathematical reasoning - in a *formal language*.

Alas, description of such systems is not an easy task. The most famous example is the discovery in 1912, by Russel, of a *paradox* (i.e. the ability to prove in a given system any property expressible in it) in Frege's logical system. Such a logical system is said to be *inconsistent*. It was then clear that studying these systems was necessary: meta-mathematics was born.

Since then, numerous logical systems has been defined, up to the point that, in 1920, Hilbert formulated his so-called *Hilbert's Program*, which goal was to formalize modern mathematics in a logical system, and to provide a proof that this system is not inconsistent. The incompleteness theorem of Gödel [27] partially answered negatively to Hilbert's Program.

The ability of computers to carry out complex computations operating on symbolic expressions renewed the area of formal mathematics: the mechanization of *proof-checking*, as well as the automatic search for proofs was born. On the practical side, *theorem provers* have been developed since then by computer scientists, and become now used as tools for developing formal mathematical proofs.

The first proof-checking system was developed in the 60's by N.G. Bruijn, under the name of AUTOMATH [17], a formal language allowing for the description and verification of mathematical proofs. In AUTOMATH, the user had to enter directly the proof of the proposition being proved, without help of the system.

The next major step is due to Robin Milner: *proof assistants* such as LCF [33] allow to build the proof of a proposition by applying *proof tactics*, generating a proof term that can be checked with respect to the rules of the underlying logic. The proof-checker, also called *kernel* of the proof assistant, implements the inference and deduction rules of the logic on top of a term manipulation layer.

Since then, numerous proofs assistants has been developed, including Coq [10], Nu-

Prl [9], PVS [36, 42], HOL [26], Isabelle [37, 38], Mizar [47], ..., among many others.

Why so many? Despite the immense progress made since AUTHOMATH and LCF, developing a formal proof is still quite painful. Only experts can use these systems, and they can only carry out small developments in a reasonable amount of time. Since these systems have already been proved very useful for practical applications, the need for scaling up their abilities has become a major urgent problem.

It is commonly agreed that the success of future proof assistants will rely on their ability to incorporate computations within deduction in order to mimic the mathematician when replacing the proof of a proposition P by the proof of a proposition P' obtained from P thanks to possibly complex computations.

Our goal in this thesis is to scale up the abilities of the system Coq. The intuitionist logic on which Coq is based on is the Calculus of Constructions (CC) of Coquand and Huet [13], an impredicative type theory incorporating polymorphism, dependent types and type constructors. As other logics, CC enjoys a computation mechanism called *cut-elimination*, which is nothing but the β-reduction rule of the underlying λ-calculus. But unlike others, CC enjoys a powerful type-checking rule, called *conversion*, which incorporates computations within deduction.

The traditional view that computations coincide with β-reduction suffers several drawbacks. A methodological one is that the user must encode other forms of computations as deductions, which is usually done by using ad-hoc, complex tactics. A practical one is that proofs become much larger than necessary, up to a point that they cannot be type-checked anymore. These questions become extremely important when carrying out complex developments involving large amounts of computation as the formal proof of the Four Color Theorem completed by G. Gonthier and B. Werner using Coq [25].

The Calculus of Inductive Constructions of T. Coquand and C. Paulin was a first attempt to solve this problem by introducing inductive types and the associated elimination principle rules [15]. The recent versions of Coq are based on a slight generalization of this calculus [22].

A more general attempt has been carried out since the early 90's by adding user-defined computations as rewrite rules, resulting in the Calculus of Algebraic Constructions [4]. Although conceptually quite powerful, since CAC captures CIC [5], this paradigm does not yet fulfill all needs, because the set of user-defined rewrite rules must satisfy several strong assumptions. As of today, no implementation of CAC has been released.

Besides, in 1998, G. Dowek, T. Hardin and C. Kirchner [18] proposed a new system for combining deduction with first-order logic: the *Natural Deduction Modulo*. This system is an extension of Natural Deduction [39] where all rules are applied modulo a congruence on propositions. In [19], D. Dowek and B. Werner gave general conditions for ensuring the termination of cut elimination in Natural Deduction Modulo. In [18], they proved that arithmetic can be presented as a theory modulo in such a way that cut elimination holds.

The proof assistant PVS uses a potentially stronger paradigm than Coq by combining its deduction mechanism[1] with a notion of computation based on the powerful Shostak's method for combining first order decision procedures over equality [43], a framework

[1] PVS logic is not based on the Curry-Howard principle and proof-checking is not even decidable, making both frameworks very different and difficult to compare.

dubbed *little engines of proof* by N. Shankar [41]: the little engines of proof are the decision procedures combined by Shostak's algorithm.

In this thesis, we investigate a new version of the Calculus of Inductive Constructions which incorporates arbitrary decision procedures into deduction via the conversion rule of the calculus.

1.1 A brief history of type theory

We introduce here some basic concepts about type theories, λ-calculi, [...] via an informal historical summary. Readers familiar with these notions, notably with Pure Type Systems, can skip this section.

An extended introduction, from which this section is inspired, can be found in the course notes of Gilles Dowek [2].

Lambda calculus

In 1930, Church introduced the λ-calculus, a formalism for representing functions. Roughly, in λ-calculus, we have

- objects which are either variables, λ-abstractions $\lambda[x].t$ (representing the function associating t to the variable x), and applications of two objects $t\,u$.
- an axiom, called β-convertibility:
$$(\lambda x_A.t)\,u = t\{x \mapsto u\}$$
where $t\{x \mapsto u\}$ stands for t where all the occurrences of x has been replaced by u.

The λ-calculus is a powerful formalism. Indeed, using Church encoding of natural numbers, it is possible to express in λ-calculus all the computable functions over natural numbers. However, when seen as a logical system, it is possible to encode the Russel paradox in λ-calculus.

The idea of the *simple type theory* originally by Whitehead and Russell, and later elaborated by Church [8], was to restrict the application of objects so that the Russel paradox can not be expressed anymore.

Simple Type Theory

A simple type is either the constant ι (for base objects), the constant o (for propositions), or the functional type $A \to B$ ($\to$ being right associative) where A and B are simple types.

For example, if ι represents the type of natural numbers, the following types are given to the symbols of arithmetic: $0 : \iota$, $S : \iota \to \iota$, $+ : \iota \to \iota \to \iota$, whereas the equality over natural numbers has type $\iota \to \iota \to o$.

More generally, in the λ-calculus presentation of the simple type theory (called *simply typed* λ-*calculus*), are given: i) for each simple type A, an infinite set $\mathcal{X}^A$ of variables of type

[2] http://www.lix.polytechnique.fr/~dowek/Cours/theories_des_types.ps.gz

A is given, and ii) a set of constants with associated types: $\overline{\vee} : o \to o \to o$, $\overline{\wedge} : o \to o \to o$, $=_A : A \to A \to A$, $\forall_A : (A \to o) \to o$, etc...

The terms of the theory are then the simply typed λ-terms, where i) variables and constants of type A are terms of type A, ii) if t is a term of type B and x a term of type A, then $\lambda x.\, t$ is a term of type $A \to B$, iii) if t is a term of type $A \to B$ and u a term of type A, then $t\,u$ is a term of type B.

Now, if ι represents the set of natural numbers, the predicate **even** is represented by the expression $\lambda x_i.\, \exists_i (\lambda y_i.\, x_i = 2 \times y_i)$, of type $\iota \to o$.

Typing judgments

Instead of requiring an infinite set of variables for each base type, it is possible to defined a variant of the simply typed λ-calculus where variables are untyped, type annotations appear in the λ-terms and *typing contexts* are used to indicate the types of the variables.

A *pure lambda term* is then either i) a variable, ii) an abstraction $\lambda[x : A].\, t$ where A is a simple type, or iii) an application $t\,u$.

Given a *typing context* Γ, i.e. a list of pairs (x, A) where a variables do not appear twice, the judgment t is of type A under Γ, written $\Gamma \vdash t : A$, can be derived by using the following rules:

$$\frac{(x, A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B} \qquad \frac{\Gamma, (x, A) \vdash t : B}{\Gamma \vdash \lambda[x : A].\, t : A \to B}$$

Convertible propositions

In the simply typed λ-calculus, if we have a proof π of $0 =_\iota 0$ (P_1), then we can easily construct a proof ψ of $(\lambda x_i.\, x_i)\, 0 =_\iota 0$ (P_2), using convertibility of β-redexes and Leibniz axioms. But, although π and ψ are proofs of equivalent propositions, they are not identical as they differ by an explicit conversion.

A solution is to equate the propositions P_1 and P_2 so that a proof π of P_1 is also a proof of P_2. This leads to the definition of a theory *modulo* where all terms are identified up to β-convertibility. Moreover, by orienting the β-convertibility $((\lambda x.\, t)\, u \xrightarrow{\beta} t\{x \mapsto u\})$, we obtain a confluent and normalizing rewrite system, leading to a canonical representation of propositions - $0 =_\iota 0$ being the canonical form of $(\lambda x_i.\, x_i)\, 0 =_\iota 0$ in our example.

Curry-Howard isomorphism

Heyting proposed a semantics of proofs as functional objects, proofs of axioms being objects given *a priori*. For example, a proof of $A \Rightarrow B$ (resp. $\forall x.\, A$) is seen as a function from A to B (resp. a function associating to any object t a proof of $A\{x \mapsto t\}$).

Following Heyting's semantic, Curry remarked [16] that a correspondence could be drawn between types of the simply typed λ-calculus and propositions of the minimal propositional logic, and between the terms of a certain type A and the proofs of the propositions corresponding to A. Such a correspondence allows us to represent proofs of minimal propositional logic with simply typed λ-terms.

The translation $\underline{P}$ of a proposition P is simply obtained by replacing occurrences of $\Rightarrow$ by the functional type constructor $\to$ and atomic propositions A with a base type ι_A. For example, the type corresponding to the proposition $(A \Rightarrow B) \Rightarrow A \Rightarrow B$ is the type $(\iota_A \to \iota_B) \to \iota_A \to \iota_B$.

Translation of proofs follows by a direct induction. Given a proof of $A_1, \cdots, A_n \vdash B$ in natural deduction for the minimal propositional logic of Gentzen [20], we construct a λ-term of type $\underline{B}$ in the context $(x_1, \underline{A_1}), \cdots, (x_n, \underline{A_n})$:

- the proof of $A_1, \cdots, A_n \vdash A_i$ is translated to x_i.
- the proof $\Omega \vdash P \Rightarrow Q$ obtained via $\Rightarrow$-intro from the proof π of $\Omega, P \vdash Q$ is translated to $\lambda[x : \underline{P}].\, q$ where q is the translation of π.
- the proof of $\Omega \vdash Q$ obtained via $\Rightarrow$-elim from the proofs π of $\Omega \vdash P \Rightarrow Q$ and ψ of $\Omega \vdash P$ is translated to $t\,u$ where t is the translation of π and u the translation of ψ.

By denoting $\underline{\pi}$ the translation of π, it is easy to check that

$$\pi \text{ is a proof of } A_1, \cdots, A_n \vdash B \text{ if and only if } (x_1, \underline{A_i}), \cdots, (x_n, \underline{A_n}) \vdash \underline{\pi} : \underline{B}.$$

As an example, the translation of the usual proof $(A \Rightarrow B) \Rightarrow A \Rightarrow B$ is the λ-term

$$\lambda[p : \iota_A \to \iota_B][x : \iota_A].\, p\, x.$$

Cut Elimination

One important remark done by W. Tait: β-reduction coincide with the cut elimination in natural deduction. Indeed, from the following proof π containing a cut (the corresponding λ-terms are also given):

$$\dfrac{\dfrac{\Gamma, x : P \vdash t : Q}{\Gamma \vdash \lambda[x : \underline{P}].\, t : P \Rightarrow Q} \Rightarrow\text{-intro} \qquad \dfrac{}{\Gamma \vdash u : P}}{\Gamma \vdash (\lambda[x : \underline{P}].\, t)\, u : Q} \Rightarrow\text{-elim}$$

it is easy to check the term $t\{x \mapsto u\}$ (obtained by β-reduction from $(\lambda[x : \underline{P}].\, t)\, u$) is the translation of the proof obtained from π in which the cut has been eliminated.

This emphasizes the importance of the normalization of β-reduction in these systems.

Beyond minimal propositional logic

Although the simply typed λ-calculus is powerful enough to represent any proof of the minimal propositional logic, it is not powerful enough to represent the proofs of all the simply type logic or the first order logic. Some extension has then be done in order to capture more proofs.

Dependent types

N. de Bruijn and Howard introduced in 1968 [17] and 1969 [28] the notion of *dependent types*, an extension of simple types which permits to capture all propositions and proofs of intuitionist first-order logic.

A dependent type is simply a function from objects to types. A canonical example of dependent types is the type of vectors where the size for the vector is present in its type. Thus, there is not a single type **vector** for vectors, but an infinite family of types **vector** 0, **vector** 1, etc... for vectors of size 0, 1, etc... One problem arises when looking the type of a function f which takes a natural number n and returns an vector of size n. The simple arrow type is not expressive enough since no dependency exists between the codomain and the domain of an arrow type. A new type constructor is therefore introduced: the *dependent product* $\forall(x : A).\, B$ which is a generalization of $A \to B$ where B can depend from x. Returning to our function f, we can now write its type as $\forall(x : \mathbf{nat}).\, \mathbf{vector}\, n$.

One important point with dependent type systems is that there is no syntactic distinction between terms and types anymore: a single algebra of terms is given, the distinction between object level terms and type level terms being done at typing. For that purpose, two new constants (called *sorts*) are introduced: the type of propositions and basic types $\star$, and the type of predicate types $\Box$.For example, the type of natural number **nat** has type $\star$, whereas the type of the **even** predicate has type $\mathbf{nat} \to \star$ which itself has type $\Box$.

As for simply typed λ-calculus, a notion of typing judgments was given for the dependent λ-calculus, also called $\lambda\Pi$-calculus. As an example, we give the rules for the formation of dependent products and sorts in $\lambda\Pi$

$$\frac{\Gamma \vdash T : \star \quad \Gamma, [x : T] \vdash U : s \in \{\star, \Box\}}{\Gamma \vdash \forall(x : T).\, U : s} \qquad \frac{}{\vdash \star : \Box}$$

Such a rule allows e.g. the formation of the type $\mathbf{nat} \to \star$ (for typing the **vector** constructor e.g.), or the type $\mathbf{nat} \to \Box$ (for typing the proposition $\forall(n : \mathbf{nat}).\, \mathbf{even}\, n$ e.g.).

Rules for application and λ-abstraction are then generalized so that they take into account the dependency of products (hence the substitution in the type of an application):

$$\frac{\Gamma, [x : T] \vdash u : U \quad \Gamma \vdash \forall(x : T).\, U : s \in \{\star, \Box\}}{\Gamma \vdash \lambda[x : T].\, u : \forall(x : T).\, U} \qquad \frac{\Gamma \vdash t : \forall(x : U).\, V \quad \Gamma \vdash u : U}{\Gamma \vdash t\, u : V\{x \mapsto u\}}$$

Last, as stated in previous section, β-convertible propositions are identified. This is done in $\lambda\Pi$ by the introduction of a *conversion* typing rule:

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s \in \{\star, \Box\} \quad T \overset{\beta}{\longleftrightarrow}_* T'}{\Gamma \vdash t : T'}$$

Polymorphism

Although $\lambda\Pi$-calculus captures more proof than simply typed λ-calculus, it does not capture the *impredicative* part of simple type theory, i.e. the ability to quantify over predicates.

This problem was solved by J.-Y. Girard [23, 24] with its polymorphic λ-calculus: $F\omega$. What forbids us to quantify over predicates in $\lambda\Pi$ is the restriction over products domain: they must be of type $\star$. This restriction is removed in $F\omega$, the new rules for products construction being:

$$\frac{\Gamma \vdash T : \Box \quad \Gamma, [x : T] \vdash U : s \in \{\star, \Box\}}{\Gamma \vdash \forall(x : T).\, U : s} \qquad \frac{\Gamma \vdash T : \star \quad \Gamma, [x : T] \vdash U : \star}{\Gamma \vdash \forall(x : T).\, U : \star}$$

For example, in such a system, it is possible to type the term $\forall(P : \star).\, P \to P$ (i.e. the type corresponding to the proposition $\forall P : o.\, P \Rightarrow P$ in simply typed logic) as $\square \to \star$ is a valid type. Likewise, it is possible to define type constructors, like the $\mathbf{list} : \star \to \star$ where $\mathbf{list}\, A$ denotes the lists whose element are of type A.

Calculus of Constructions

In 1991, H. Barendregt remarks that all these typed λ-calculi differ from their products formation typing rules, being of the form:

$$\frac{\Gamma \vdash T : s_1 \quad ,\Gamma, [T : s_1] \vdash U : s_2}{\Gamma \vdash \forall(x : T).\, U : s_2}$$

where s_1, s_2 are sorts. For example, in the simply typed calculus, $s_1 = s_2 = \star$ whereas in $\lambda\Pi$, $s_1 = \star$ and $s_2 \in \{\star, \square\}$.

The Calculus of Constructions (CC) of T. Coquand and G. Huet [14], which is the basis of the Coq proof assistant, is simply obtained by allowing all kinds of products (i.e. s_1 and s_2 being unrestricted).

Logical Systems Expressiveness

One can ask which functions are expressible using β-reduction of typed λ-calculi: very few indeed. For example, in the simply typed λ-calculus, using Peano representation of natural numbers (i.e. using a base type $\mathbf{nat}$ and two symbols $\mathbf{0} : \mathbf{nat}$ and $\mathbf{S} : \mathbf{nat} \to \mathbf{nat}$), it is only possible to express constant functions and functions adding a constant to their arguments.

On the contrary, in simple type theory, it is possible to prove the existence of functions that are not expressible using λ-terms: while it is possible to construct proofs using the induction principle, it is not possible to β-define functions using recursion.

Inductive Types

This is the idea of Gödel system T: extending the simply typed λ-calculus so that such definitions are possible. For this, besides the constant $\mathbf{nat}$, $\mathbf{0}$ and $\mathbf{S}$, a new symbol Elim^T of type $T \to (\mathbf{nat} \to T \to T) \to \mathbf{nat} \to T$ is introduced for every type T, along with new reduction rules:

$$\mathrm{Elim}^T\ v_0\, v_S\, \mathbf{0} \xrightarrow{\iota} v_0 \qquad \mathrm{Elim}^T\ v_0\, v_S\, (\mathbf{S}\, t) \xrightarrow{\iota} v_S\, t\, (\mathrm{Elim}\ v_0\, v_S\, t)$$

In such a system, addition can then simply defined as the λ-term

$$\lambda[x\, y : \mathbf{nat}].\, \mathrm{Elim}^{\mathbf{nat}}\ y\, (\lambda[r\, z : \mathbf{nat}].\, \mathbf{S}\, z)\, x$$

In the Calculus of Constructions, the problem is different. Although it is possible to express far more functions than in the simply typed λ-calculus, one can not express functions as she likes due to the encoding used for the embedding of natural numbers (we say that natural numbers are impredicatively defined). For example, it is not possible to define the predecessor function so that its evaluation is in constant time.

This is the main reason for the introduction by T. Coquand and C. Paulin [15] of the Calculus of Inductive Constructions (CIC), an extension of CC where it is possible

to define inductive types and used their induction principles to define terms. As for the Gödel's system T, new reduction rules (the ι-reduction) are added for the elimination of recursor symbols. Moreover, following P. Martin-Löf [31] type theory (an extension of λΠ with inductive types), the ι-reduction is added to the conversion rule.

One point of having ι-reduction in the conversion is that more propositions are identified. E.g., if + is defined by induction on its first argument and P is a predicate over natural numbers, then the two propositions $P\,x$ and $P\,(\mathbf{0}+x)$ are identified since these two terms are convertible. Alas, this is not true for $P\,x$ and $P\,(x+\mathbf{0})$, $x+\mathbf{0}$ being not convertible to x by ι-reduction.

Rewriting in the Calculus of Constructions

One first solution to this problem is the introduction in the conversion rule of a rewriting system along with the standard β-reduction. Lot of works has been done along this, the most noticeable one being the Calculus of Algebraic Constructions of F. Blanqui [4].

As an example, using rewriting, one can define addition as the following rewriting system

$$\mathbf{0}+x \to x \quad x+\mathbf{0} \to x \quad (\mathbf{S}\,x)+y \to \mathbf{S}(x+y).$$

The immediate consequence is the identification of the three propositions $P\,x$, $P\,(x+\mathbf{0})$ and $P\,(\mathbf{0}+x)$.

Extensionality in the Calculus of Constructions

One more recent extension is the Extensional Calculus of Constructions (CC_E) of N. Oury [35] which is an extension of the Extended Calculus of Constructions (a Pure Type System with a hierarchy of cumulative sorts $\square_i$ and an *impredicative* sort $\star$ [30]).

Roughly, terms and typing rules of CC_E are as in the Extended Calculus of Constructions, but with an abstract conversion relation $\equiv_\Gamma$ instead of the standard β-conversion. The relation $\equiv_\Gamma$ is then defined as the weakest congruence including β-conversion and the propositional Leibniz equality of CC_E (denoted by $\doteq$), hence the name of *Extensional* Calculus of Constructions. (Rules of Figure 1.1 define the Conversion $\equiv_\Gamma$ where $\vdash_E$ denotes the typing judgment of CC_E - the reflexivity, symmetry and transitivity rules being omitted)

$$\frac{\Gamma \vdash_E (\lambda[x:U].\,t)\,u : T}{\lambda[x:U].\,t \equiv_\Gamma t\{x \mapsto u\}}\ [\beta] \qquad \frac{t_1 \equiv_\Gamma u_1 \quad t_2 \equiv_\Gamma u_2}{t_1\,t_2 \equiv_\Gamma u_1\,u_2}\ [\textsc{App}]$$

$$\frac{T \equiv_\Gamma U \quad t \equiv_{\Gamma,[x:T]} u}{\forall(x:T).\,t \equiv_\Gamma \forall(x:U).\,u}\ [\textsc{Prod}] \qquad \frac{T \equiv_\Gamma U \quad t \equiv_{\Gamma,[x:T]} u}{\forall(x:T).\,t \equiv_\Gamma \forall(x:U).\,u}\ [\textsc{Lam}]$$

$$\frac{\Gamma \vdash p : T_1 = T_2}{T_1 \equiv_\Gamma T_2}\ [\textsc{Ext}]$$

Figure 1.1: CC_E Conversion Relation

As shown in [35], this calculus is expressive enough to convert two functions which are point to point equal. Indeed, if in a typing environment $\Gamma, [x : A]$, we have a proof of $t \doteq u$, then we easily obtain that $\lambda[x : A].\, t$ is convertible to $\lambda[x : A].\, u$:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, [x : A] \vdash_E p : t \doteq u}\,[\textsc{Ext}]}{t \equiv_{\Gamma,[x:A]} u} \qquad \dfrac{}{A \equiv_\Gamma A}\,[\textsc{Refl}]}{\lambda[x : A].\, t \equiv_\Gamma \lambda[x : A].\, u}\,[\textsc{Lem}]$$

Unfortunately, such a powerful conversion rule immediately leads to the undecidability of type-checking. Intuitively, this is due to the [Ext] rule erasing the witness of equality properties. See [35] for an encoding of the halting problem into the CC_E type-checking problem.

Worst, basic properties of Pure Type Systems are lost. For example, β-strong normalization of well-formed terms is lost as the whole lambda calculus can be encoded in a context Γ containing an equation of the form $A \doteq A \to A$. Using extensionality, it is immediate to verify that the identity $id \equiv \lambda[x : A].\, x$ over A has type $A \to (A \to A)$ (roll) and type $(A \to A) \to A$ (unroll). Thus, the standard encoding $|\cdot|$ of pure lambda calculus to the simply typed lambda calculus with recursive types can be used:

$$|x| = x,\ |\lambda x.\, t| = \mathrm{roll}\,(\lambda[x : A].\, |t|),\ |m\, n| = (\mathrm{unroll}\,|m|)\,|n|$$

One can then check that any closed term $|t|$ is well formed under Γ.

1.2 Safety of Proof assistants

The safety of proof assistants is based on the trustability of their kernel, a proof-checker that processes all proofs built by a user with the help of tactics that are available from existing libraries or can otherwise be developed for achieving a specific task. In the early days of Coq, the safety of its proof-checker relied on its small size and its clear structure reflecting the inference rules of the intuitionistic type theory, the Calculus of Constructions, on which it was based. The slogan was that of a *readable kernel*.

Moving later to the Calculus of Inductive Constructions allowed to ease the specification tasks, making the system very popular among proof developers, but resulted in a more complex kernel that can now hardly be read except by a few specialists. The slogan changed to a *provable kernel*, and indeed, one version of Coq kernel was once proved with an earlier version (using strong normalization as assumption), and a new safe kernel was extracted from that proof [1].

Of course, there has been many changes in the kernel since then, and its correctness proof was of course not maintained. This is a first weakness with the *provable kernel* paradigm: it does not resist changes very well. There is a second, more important, which relates directly to our calculus: there is no guarantee that a decision procedure taken from the shelf implements correctly the complex mathematical theorem on which it is based, since carrying out such a proof may require an entire PhD work. Therefore, these procedures *cannot* be part of the kernel, and be used to identify propositions in the conversion relation.

1.3 Contributions

Our main contribution is the definition and study of a new calculus, the Calculus of Congruent and Inductive Constructions, an extension of the Calculus of Inductive Constructions (CIC) integrating in its computational part the entailment relation of a first order theory over equality. A major technical innovation of this work lies in the computational mechanism: goals are sent to the decision procedure together with a set of user hypotheses available from the current context.

Our main results show that this extension of CIC does not compromise its main properties: confluence, strong normalization, consistency and decidability of proof-checking are all preserved (as soon as the incorporated theory is itself decidable). As such, our calculus can be seen as a decidable restriction of the Extensional Calculus of Constructions. It can therefore serve as the basis for an extension of the Coq proof assistant.

Unlike previous calculi, the main difficulty here is confluence, which led to a complex definition of conversion as a fix-point. As a consequence, decidability of type-checking becomes itself difficult, and does not reduce to the problem of terms reduction w.r.t. a rewriting system. Instead, a new decision algorithm, mixing standard CIC reduction and a saturation algorithm w.r.t. the incorporated theory, is defined.

1.4 Outline of the thesis

The document is structured as follow:

Chapter 2 *The Calculus of Presburger Constructions.* We define the Calculus of Presburger Constructions, an extension of the Calculus of Inductive Constructions (without strong reduction) integrating in its computational part the Presburger arithmetic, and describe how its conversion relation can be decide.

Chapter 3 *The Calculus of Congruent and Inductive Constructions.* We define our main Calculus, the Calculus of Inductive and Congruent Constructions, and prove all its meta-theoretical properties but the decidability.

Chapter 4 *Meta-theoretical Properties of CCIC* . We state and prove all the needed meta-theoretical properties of CCIC.

Chapter 5 *Decidability.* We here describe how our calculus can be decide and give proofs of completeness and correction of the procedure.

Chapter 6 *Further works and conclusion* We conclude by enumerating several directions for future research.

CHAPTER 2

The Calculus of Presburger Inductive Constructions

Before describing our calculus in all its generality, we first define in this chapter $CC_{\mathbb{N}}$ [6], an extension of the calculus of constructions incorporating i) a type **nat** of natural numbers generated by its two constructors **0** and **S**, and equipped with its addition + and weak recursor $Rec_{\mathbb{N}}^{\mathcal{W}}$, ii) a polymorphic equality symbol $\doteq$.

The main modification is obtained by replacing $T \overset{\beta}{\longleftrightarrow}_* T'$ in the conversion rule:

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s' \quad T \overset{\beta}{\longleftrightarrow}_* T'}{\Gamma \vdash t : T'}$$

by a contextual relation $T \sim_\Gamma T'$ containing (in addition to β-reduction and the reduction associated with $Rec_{\mathbb{N}}^{\mathcal{W}}$) the entailment relation $\models$ of Presburger arithmetic $\mathcal{T}_{\mathbb{N}}$. For example, $\sim_\Gamma$ will include the property $\forall n.\, \forall p.\, n + p \sim_\Gamma p + n$, since $\mathcal{T}_{\mathbb{N}} \models n + p = p + n$ holds. The relation $\sim_\Gamma$ will also includes any $\mathcal{T}_{\mathbb{N}}$-equation *extractable* from the typing environment Γ s.t., if Γ contains - for example - the two equations $n = p + 1$ and $p = 1$, then $n \sim_\Gamma 2$ will hold (as $\mathcal{T}_{\mathbb{N}} \models (n = p + 1 \wedge p = 1) \Rightarrow n = 2$).

See Section 2.3 for detailed examples.

From now on, $\mathcal{T}_{\mathbb{N}}$ denotes the Presburger arithmetic over the signature $\Sigma = \{0, S, +\}$ and equality predicate $=$. We write $\mathcal{T}_{\mathbb{N}} \models P$ if P is a valid $\mathcal{T}_{\mathbb{N}}$-formula and $\mathcal{T}_{\mathbb{N}}, E \models P$ if $\mathcal{T}_{\mathbb{N}} \models \bigwedge\{Q \mid Q \in F\} \Rightarrow P$ for some possibly infinite set E of $\mathcal{T}_{\mathbb{N}}$-formulas and some finite subset F of E. For any set of variables $\mathcal{Y}$, $\mathcal{T}_\Sigma(\mathcal{Y})$ denotes the set of $\mathcal{T}_{\mathbb{N}}$-terms over variables $\mathcal{Y}$. We write $\mathcal{T}_\Sigma$ for $\mathcal{T}_\Sigma(\varnothing)$.

2.1 Terms of the calculus

$CC_{\mathbb{N}}$ uses two *sorts*: $\star$ (or Prop, or *object level universe*) and $\square$ (or Type, or *predicate level universe*). We denote the set $\{\star, \square\}$ of $CC_{\mathbb{N}}$ sorts by $\mathcal{S}$.

As usual, following the presentation of *Pure Type Systems* [21], we use two classes of variables: let $\mathcal{X}^\star$ (resp. $\mathcal{X}^\square$) a countably infinite set of *term variables* (resp. *predicate variables*) such that $\mathcal{X}^\star$ and $\mathcal{X}^\square$ are disjoint. We write $\mathcal{X}$ for $\mathcal{X}^\star \cup \mathcal{X}^\square$. If $x \in \mathcal{X}^s$, we write s_x for s.

Let $\mathcal{A} = \{r, u\}$ be a set of two constants, called *annotations*, where r stands for *restricted* and u for *unrestricted*.

We use the following notations:

s	range over	$\mathcal{S}$
$x, y, \ldots$	–	$\mathcal{X}$
$X, Y, \ldots$	–	$\mathcal{X}^\square$
$a, b, \ldots$	–	$\mathcal{A}$

Definition 2.1 — **$CC_{\mathbb{N}}$ terms algebra**

The algebra $CC_{\mathbb{N}}$ of pseudo-terms *of $CC_{\mathbb{N}}$ is defined as:*

$$\begin{aligned} t, u, T, U, \ldots := s \in \mathcal{S} \mid x \in \mathcal{X} \mid \mathbf{nat} \mid \mathbf{0} \mid \mathbf{S} \mid \dot{+} \mid \mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(t, U)\{v_0, v_S\} \\ \mid \forall(x :^a T).\, t \mid \lambda[x :^a T].\, t \mid t\,u \mid \doteq \mid \mathrm{Eq}_T(t) \mid \mathrm{Leib} \end{aligned}$$

Note 1

Apart from the introduction of new symbols for natural numbers and the equality predicate, the difference between these terms and the ones from the Calculus of Constructions is the introduction of annotations for products and abstractions.

Notation. The polymorphic equality symbol will be used in *mixfix* form $t \doteq_T u$ or $t \doteq u$ when T is not relevant. $\mathrm{Eq}_T(t)$ will denote the *proof by reflexivity* of $t \doteq_T t$ and Leib the Leibniz equality predicate. We shall distinguish the first-order equality predicate $=$ from the $CC_{\mathbb{N}}$ polymorphic equality $\doteq$. We also shall distinguish the first order symbols (0, S and $+$) of $\mathcal{T}_{\mathbb{N}}$ from their $CC_{\mathbb{N}}$ counterpart ($\mathbf{0}$, $\mathbf{S}$ and $\dot{+}$). We write $\mathbf{p}$ (in bold face) for the p-iteration $\mathbf{S}\,(\cdots\,(\mathbf{S}\,\mathbf{0}))$ and p (in normal font) for the p-iteration $S(\cdots S(0))$.

Note 2

The notion of free variables is as usual. If t is a $CC_{\mathbb{N}}$ term, we write $\mathrm{FV}(t)$ for the set of free variables of t. We say that t is closed if $\mathrm{FV}(t) = \varnothing$. A variable x *occurs freely* in t if $x \in \mathrm{FV}(t)$.

Note 3

If θ in a $\mathcal{L}$-*substitution* (i.e. a finite mapping $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ from $\mathcal{X}$ to $\mathcal{L}$), we write $t\theta$ for the *(capture free) substitution* of the x_i's by the u_i's in t. If $\vec{\theta_i}$ is a sequence of $\mathcal{L}$-substitutions, the *sequential substitution* $t\vec{\theta_i}$ is defined by: i) $t\epsilon = t$ ii) $t(\theta\vec{\theta_i}) = (t\theta)\vec{\theta_i}$. The *domain* (resp. *co-domain*) of a $\mathcal{L}$- substitution θ is written $\mathrm{dom}(\theta)$ (resp. $\mathrm{codom}(\theta)$).

2.2 The conversion relation

Our calculus has a complex notion of computation reflecting its rich structure made of three ingredients: the typed lambda calculus, the type of natural numbers with its recursor and the integration of the first order theory $\mathcal{T}_{\mathbb{N}}$ in its conversion.

Inductive definitions

A definition by induction over natural numbers is denoted by the dependent recursor *a la* Martin-Löf $\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(n, T)\{t_0, t_S\}$ where t is the term being deconstructed, $T\,t$ the type of the result, and t_0, t_S the branches of the definition. The reduction relation associated to $\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}$ is defined as usual:

Definition 2.2 — **ι-reduction**

The ι-reduction $\xrightarrow{\iota}$ is the smallest rewriting relation s.t.:

$$\begin{aligned} \mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(\mathbf{0}, T)\{t_0, t_S\} &\xrightarrow{\iota} t_0 \\ \mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(\mathbf{S}\,t, T)\{t_0, t_S\} &\xrightarrow{\iota} t_S\,t\;\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(t, T)\{t_0, t_S\} \end{aligned}$$

A canonical example of inductive definitions over natural numbers is the definition of the addition or multiplication (here multiplication, since addition has been internalized) by induction:

$$\lambda[x\,y : \mathbf{nat}].\,\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(x, \lambda[n : \mathbf{nat}].\,\mathbf{nat})\{\mathbf{0}, \lambda[p\,r : \mathbf{nat}].\,r \dot{+} y\}$$

Algebraisation

As said, our conversion sends goals to the Presburger arithmetic $\mathcal{T}_{\mathbb{N}}$ along with a set of proof hypotheses extracted from the typing environment. Algebraisation is the first step of this hypotheses extraction: it allows transforming a $\mathrm{CC}_{\mathbb{N}}$ term into its first-order counterpart.

We begin by the simplest case, the extraction of pure algebraic equations. Suppose that the proof environment contains equations of the form $c \doteq 1 \dot{+} d$ and $d \doteq 2$. What is expected is that the set of hypotheses sent to the theory $\mathcal{T}_{\mathbb{N}}$ contains the two well formed $\mathcal{T}_{\mathbb{N}}$-formulas $c = 1 + d$ and $d = 2$. This leads to a first definition of equations extraction:

1. a term is algebraic if it is of the form $\mathbf{0}$, or $\mathbf{S}\,t$, or $t \dot{+} u$, or $x \in \mathcal{X}^\star$. The *algebraisation* $\mathcal{A}(t)$ of an algebraic term is then defined by induction: $\mathcal{A}(\mathbf{0}) = 0$, $\mathcal{A}(\mathbf{S}\,t) = S(\mathcal{A}(t))$, $\mathcal{A}(t \dot{+} u) = \mathcal{A}(t) + \mathcal{A}(u)$ and $\mathcal{A}(x) = x$,
2. a term is an extractable equation if it is of the form $t \doteq u$ where t and u are algebraic terms. The extracted equation is then $\mathcal{A}(t) = \mathcal{A}(u)$.

We now come to the case of algebraisation of non-pure algebraic terms or even ill-formed terms. The problem can be simply solved by abstracting non-algebraic subterms with fresh variables. For example, algebraisation of $1 \dot{+} t$ with t non-algebraic will yield $1 + x$ where x is an abstraction variable. Of course, if the proof context contains two equations of the form $c \doteq 1 \dot{+} t$ and $d \doteq 1 \dot{+} u$ with t and u β-convertible, t and u should be abstracted by a unique variable so that $c = d$ can be deduced in $\mathcal{T}_{\mathbb{N}}$ from $c = 1 + y$ and $d = 1 + y$.

We now give the formal definition of $\mathcal{A}(\cdot)$.

Let $\mathcal{Y}$ be a set of variables disjoint from $\mathcal{X}$. For any equivalence relation $\mathcal{R}$, we suppose the existence of a function $\pi_{\mathcal{R}} : \mathrm{CC}_{\mathbb{N}} \to \mathcal{Y}$ s.t. $\pi_{\mathcal{R}}(t) = \pi_{\mathcal{R}}(u)$ if and only if $t\ \mathcal{R}\ u$ (i.e. $\pi_{\mathcal{R}}(t)$ is the variable in $\mathcal{Y}$ representing the class of t modulo $\mathcal{R}$).

Definition 2.3 — **Algebraisation**

Let t be a term in $CC_{\mathbb{N}}$ and $\mathcal{R}$ an equivalence relation. The algebraisation of t modulo $\mathcal{R}$ *is the function $\mathcal{A}_{\mathcal{R}} : CC_{\mathbb{N}} \to \mathcal{T}_{\mathbb{N}}(\mathcal{X}^\star \cup \mathcal{Y})$ defined by:*

$$\begin{aligned}
\mathcal{A}_{\mathcal{R}}(x) &= x && \text{if } x \in \mathcal{X}^\star \\
\mathcal{A}_{\mathcal{R}}(\mathbf{0}) &= 0 \\
\mathcal{A}_{\mathcal{R}}(\mathbf{S}\,t) &= S(\mathcal{A}_{\mathcal{R}}(t)) \\
\mathcal{A}_{\mathcal{R}}(t \dot{+} u) &= \mathcal{A}_{\mathcal{R}}(t) + \mathcal{A}_{\mathcal{R}}(u) \\
\mathcal{A}_{\mathcal{R}}(t) &= \Pi_{\mathcal{R}}(t) && \text{otherwise}
\end{aligned}$$

For an arbitrary relation R, $\mathcal{A}_{\mathsf{R}}$ is defined as $\mathcal{A}_{\mathcal{R}}$ where $\mathcal{R}$ is the smallest equivalence relation containing R. We call alien *the subterms of t abstracted by a variable in $\mathcal{Y}$.*

Typing environments

Conversion being contextual, we now need to define *typing environments* of our calculus.

Definition 2.4 — **Pseudo-contexts of $\mathrm{CC}_{\mathbb{N}}$**

The typing environments of $CC_{\mathbb{N}}$ are defined as $\Gamma, \Delta ::= [] \mid \Gamma, [x :^{a} T]$ *s.t. a variable cannot appear twice. We use* $\mathrm{dom}(\Gamma)$ *for the domain of* Γ *and* $x\Gamma$ *for the type associated to* x *in* Γ.

Remark that in our calculus, assumptions stored in the proof context always come along with an annotation $a \in \mathcal{A}$ used to control whether they can be used (when $a = r$) or not (when $a = u$) in a conversion goal. We will later point out why this is necessary.

Conversion relation

Befored defining our conversion relation, we are left to give the usual layered definition of PTSs terms, extended to the $\mathrm{CC}_{\mathbb{N}}$ case.

Definition 2.5 — **Syntactic classes**

The pairwise disjoint syntactic classes of $CC_{\mathbb{N}}$ called objects *($\mathcal{O}$),* predicates *($\mathcal{P}$),* kinds *($\mathcal{K}$),* $\square$ *are defined in Figure 2.1.*

$$
\begin{aligned}
\mathcal{O} ::= {} & \mathcal{X}^{\star} \mid \mathbf{0} \mid \mathbf{S} \mid \dot{+} \mid \mathcal{O}\,\mathcal{O} \mid \mathcal{O}\,\mathcal{P} \mid \mathrm{Eq}_{\mathcal{P}}(\mathcal{O}) \mid \mathrm{Leib} \\
::= {} & \lambda[x^{\star} :^{a} \mathcal{P}].\,\mathcal{O} \mid \lambda[x^{\square} :^{a} \mathcal{K}].\,\mathcal{O} \mid \mathrm{Rec}^{\mathcal{W}}_{\mathbb{N}}(\mathcal{O}, \mathcal{P})\{\mathcal{O}, \mathcal{O}\} \\
\mathcal{P} ::= {} & \mathcal{X}^{\square} \mid \mathbf{nat} \mid \doteq \mid \mathcal{P}\,\mathcal{O} \mid \mathcal{P}\,\mathcal{P} \mid \lambda[x^{\star} :^{a} \mathcal{P}].\,\mathcal{P} \mid \lambda[x^{\square} :^{a} \mathcal{K}].\,\mathcal{P} \\
::= {} & \forall(x^{\star} :^{a} \mathcal{P}).\,\mathcal{P} \mid \forall(x^{\square} :^{a} \mathcal{K}).\,\mathcal{P} \\
\mathcal{K} ::= {} & \star \mid \forall(x^{\star} :^{a} \mathcal{P}).\,\mathcal{K} \mid \forall(x^{\square} :^{a} \mathcal{K}).\,\mathcal{K} \\
\square ::= {} & \square
\end{aligned}
$$

Figure 2.1: $\mathrm{CC}_{\mathbb{N}}$ terms classes

This classes play a crucial role as we only authorize equations extraction and conversion using Presburger arithmetic to occur at object level. See Section 2.5 for a discussion about equations extraction.

We can now define the Γ-indexed family of conversion relations $\{\sim_{\Gamma}\}_{\Gamma}$.

Definition 2.6 — **Conversion relation $\sim_{\Gamma}$**

Rules of Figure 2.2 defines a family $\{\sim_{\Gamma}\}$ of $CC_{\mathbb{N}}$ binary relations.

This definition is technically complex:

- being a congruence, $\sim_{\Gamma}$ includes congruence rules. However, all these rules are not quite congruence rules since crossing a binder increases the current context Γ by the

$$\frac{t \xrightarrow{\beta\iota}_* t' \quad t' \sim_\Gamma u}{t \sim_\Gamma u}\ [\beta\iota\text{-Left}] \qquad \frac{u \xrightarrow{\beta\iota}_* u' \quad t \sim_\Gamma u'}{t \sim_\Gamma u}\ [\beta\iota\text{-Right}]$$

$$\frac{[x :^r T] \in \Gamma \quad T \xrightarrow{\beta}_* t_1 \doteq t_2 \quad t_1, t_2 \in \mathcal{O}}{t_1 \sim_\Gamma t_2}\ [\text{Eq}]$$

$$\frac{\mathcal{T}_\mathbb{N}, E \vDash \mathcal{A}_{\sim_\Gamma}(t_1) = \mathcal{A}_{\sim_\Gamma}(t_2) \quad t_1, t_2 \in \mathcal{O} \quad E = \{\mathcal{A}_{\sim_\Gamma}(u_1) = \mathcal{A}_{\sim_\Gamma}(u_2) \mid u_1 \sim_\Gamma u_2\}}{t_1 \sim_\Gamma t_2}\ [\text{Ded}]$$

$$\frac{t \sim_\Gamma u}{u \sim_\Gamma t}\ [\text{Sym}] \qquad \frac{t \sim_\Gamma u \quad u \sim_\Gamma v}{t \sim_\Gamma v}\ [\text{Trans}]$$

$$\frac{T \sim_\Gamma U \quad t \sim_\Gamma u}{\text{Eq}_T(t) \sim_\Gamma \text{Eq}_U(u)}\ [\text{CC}_\mathbb{N}\text{-Eq}] \qquad \frac{t_1 \sim_\Gamma t_2 \quad u_1 \sim_\Gamma u_2}{t_1\, u_1 \sim_\Gamma t_2\, u_2}\ [\text{App}]$$

$$\frac{T \sim_\Gamma U \quad t \sim_{\Gamma,[x:^a T]} u}{\forall(x :^a T).\, t \sim_\Gamma \forall(x :^a U).\, u}\ [\text{Prod}] \qquad \frac{T \sim_\Gamma U \quad t \sim_{\Gamma,[x:^a T]} u}{\lambda[x :^a T].\, t \sim_\Gamma \lambda[x :^a U].\, u}\ [\text{Lam}]$$

$$\frac{t \sim_\Gamma u \quad P \sim_\Gamma Q \quad t_0 \sim_\Gamma u_0 \quad t_S \sim_\Gamma u_S}{\text{Rec}^\mathcal{W}_\mathbb{N}(t, P)\{t_0, t_S\} \sim_\Gamma \text{Rec}^\mathcal{W}_\mathbb{N}(u, Q)\{u_0, u_S\}}\ [\text{Elim-}\mathcal{W}]$$

Figure 2.2: Conversion relation $\sim_\Gamma$

new assumption made inside the scope of the binding construct, resulting in a family of congruences.

- $\sim_\Gamma$ includes all the relevant assumptions grabbed from the context, this is the rule [Eq]. These assumptions must be of the form $[x :^r T]$ (i.e. with the appropriate annotation r), and T must β-reduce to a term headed by $\doteq$. Note that we do not require T to be $\sim_\Gamma$-convertible to an algebraic predicate here. Doing this would not change $\sim_\Gamma$, but would complicate the study of the calculus.
- we use the theory $\mathcal{T}_\mathbb{N}$ to generate new assumptions: this is the rule [Ded].
- Finally, $\sim_\Gamma$ includes $\beta\iota$-reductions. Unlike the β-rule, $\xrightarrow{\iota}$ interacts with first-order rewriting, and therefore forbids to express the [Conv] rule of Figure 2.4 as

$$T \xleftrightarrow{\beta\iota}_* \equiv_\Gamma \xleftrightarrow{\beta\iota}_* T'$$

($\equiv_\Gamma$ being defined like $\sim_\Gamma$ without the $\beta\iota$ rule). A simple example demonstrate this. Suppose that Γ is a typing environment containing two extractable equations $x \doteq \mathbf{0}$ and $y \doteq 1$. One can indeed easily check that

$$\text{Rec}^\mathcal{W}_\mathbb{N}(x, Q)\{y, v_S\} \equiv_\Gamma \text{Rec}^\mathcal{W}_\mathbb{N}(\mathbf{0}, Q)\{y, v_S\} \xrightarrow{\iota} y \equiv_\Gamma \mathbf{1}$$

but in general, $\text{Rec}^\mathcal{W}_\mathbb{N}(x, Q)\{y, v_S\} \equiv_\Gamma \mathbf{1}$ does not hold.

Before going to the typing rules, we give some examples of conversion.

2.3 Two simple examples

More automation - smaller proofs. We start with a simple example illustrating how the equalities extracted from a context Γ can be use to deduce new equalities in $\sim_\Gamma$.

$$\begin{aligned}\Gamma = {} & [x\ y\ t :^u \mathbf{nat}], [f :^u \mathbf{nat} \to \mathbf{nat}],\\ & [p_1 :^r t \doteq 2], [p_2 :^r f\,(x \dot{+} 3) \doteq x \dot{+} 2],\\ & [p_3 :^r f\,(y \dot{+} t) \dot{+} 2 \doteq y], [p_4 :^r y \dot{+} 1 \doteq x \dot{+} 2]\end{aligned}$$

From p_1 and p_4 (extracted from the context by [Eq]), [Ded] deduces that $y \dot{+} t \sim_\Gamma x \dot{+} 3$, and by congruence, $f\,(y \dot{+} t) \sim_\Gamma f\,(x \dot{+} 3)$. Therefore, $\pi_{\sim_\Gamma}$ abstracts $f(x \dot{+} 3)$ and $f(y \dot{+} t)$ by the same variable z, resulting in two new equations available for [Ded]: $z = x + 2$ and $z + 2 = y$. Now, $z = x + 2$, $z + 2 = y$ and $y + 1 = x + 2$ form a set of unsatisfiable equations and we deduce $0 \sim_\Gamma 1$ by the [Ded] rule: contradiction has been obtained. This shows that we can easily carry out a proof by contradiction in $\mathcal{T}_{\mathbb{N}}$.

More typable terms. We continue with a second example showing that the new calculus can type terms that are not typable in CIC. For the sake of this example we assume that $CC_{\mathbb{N}}$ is extended by dependent lists on natural numbers. These dependent lists are defined as a standard inductive type (without a built-in theory of lists). This is a simple extension of $CC_{\mathbb{N}}$ that will be justified later. We denote by **list** (of type $\mathbf{nat} \to \star$) the type of dependent lists and by **nil** (of type $\mathbf{list}\,\mathbf{0}$) and **cons** (of type $\forall(n : \mathbf{nat}).\,\mathbf{nat} \to \mathbf{list}\,n \to \mathbf{list}\,(\mathbf{S}\,n)$) the lists constructors.

Assume now given a dependent reverse function (of type $\forall(n : \mathbf{nat}).\,\mathbf{list}\,n \to \mathbf{list}\,n$) and the list concatenation function @ (of type $\forall(n\,n' : \mathbf{nat}), \mathbf{list}\,n \to \mathbf{list}\,n' \to \mathbf{list}\,(n \dot{+} n')$). We can simply express that a list l is a palindrome: l is a palindrome if reverse $l \doteq l$.

Suppose now that one wants to prove that palindromes are closed under substitution of letters by palindromes. To make it easier, we will simply consider a particular case: the list $l_1 l_2 l_2 l_1$ is a palindrome if l_1 and l_2 are palindromes. The proof sketch is simple: it suffices to apply as many times as needed the lemma $\mathrm{reverse}(l@l') = \mathrm{reverse}(l')@\,\mathrm{reverse}(l)$ $(*)$. What is quite surprising is that Lemma $(*)$ is rejected by the current version of Coq. Indeed, if l and l' are of length n and n', it is easy to check that $\mathrm{reverse}(l@l')$ is of type $\mathbf{list}\,(n \dot{+} n')$ and $\mathrm{reverse}(l')@\,\mathrm{reverse}(l)$ of type $\mathbf{list}\,(n' \dot{+} n)$ which are clearly not $\beta\iota$-convertible. This is not true in our system: $n \dot{+} n'$ will of course be convertible to $n' + n$ and lemma $(*)$ is therefore well-formed. Proving the more general property needs of course an additional induction on natural numbers to apply lemma $(*)$ the appropriate number of times, which can of course be carried out in our system.

Note that, although possible, writing a reverse function for dependent lists is not that simple in Coq. Indeed, a direct inductive definition of reverse will define $\mathrm{reverse}(\mathbf{cons}\,n\,a\,l)$, of type $\mathbf{list}\,(1 \dot{+} n)$, as $\mathrm{reverse}(l)$ @ a, of type $\mathbf{list}\,(n \dot{+} 1)$. Coq will reject such a definition since $\mathbf{list}\,(1 \dot{+} n)$ and $\mathbf{list}\,(n \dot{+} 1)$ are not convertible. Figure 2.3 shows how reverse can be (painfully) defined in Coq.

```
Coq < Definition reverse: forall (n: nat), (list n) -> (list n) .
Coq <   assert (reverse_acc : forall (n m : nat),
Coq <             list n -> list m -> list (m+n)) .
Coq <   refine (fix reverse_acc (n m : nat) (from : list n) (to : list m)
Coq <             {struct from} : list (m+n) := _) .
Coq <   destruct from as [ | n' v rest ] .
Coq <     rewrite <- plus_n_o_transparent; exact to .
Coq <     rewrite <- plus_n_Sm_transparent;
Coq <       exact (reverse_acc n' (S m) rest (cons _ v to)) .
Coq <   intros n l . exact (reverse_acc _ _ l nil) .
Coq < Defined .
```

Figure 2.3: reverse function in Coq

2.4 Typing rules

Our typing judgments are classically written $\Gamma \vdash t : T$, meaning that the *well-formed* t is a proof of the proposition T under the assumptions in the *well-formed* environment Γ. *Typing rules* are those of CIC restricted to the single inductive type of natural numbers, with two exceptions: i) the conversion rule [Conv] based on the conversion relation defined in previous section, ii) the application rule [App].

Definition 2.7 **$CC_{\mathbb{N}}$ typing relation**

Typing rules of $CC_{\mathbb{N}}$ are defined in Figure 2.4.

2.5 Consistency

We don't give a detailed proof of consistency here, (see Chapter 3 for definition of a more general calculus and for a proof of its consistency) but illustrate by examples our different design choices.

Extraction of object-level equations only

Allowing extraction of equations at type level breaks two important properties of the calculus, strong normalization of β-reduction and type convertibility:

Strong normalization of β. Assuming possible the extraction of the type level equation $\mathbf{nat} \doteq \mathbf{nat} \to \mathbf{nat}$ leads immediately to the well formation (under any environment Γ containing the extractable equation) of the non-normalizing term $\omega\,\omega$ where $\omega = \lambda[x : \mathbf{nat}].\, x\, x$.

Type convertibility. Let $(\cdot,\cdot)$ be a polymorphic constructor for pairs: $(a, b)_{(A,B)}$ is of type $A * B$ where a (resp. b) is the type of A (resp. B), and π_1 (resp. π_2) denotes the first projection (resp. the second projection) operator. Then, in any environment s.t. the equation $A * B = B * A$ is extractable, for any terms a, b of respective types A, B, $\pi_1(a, b)_{(B,A)}$ is a well typed term of type B, whereas $\pi_1(a, b)_{(B,A)} \xrightarrow{\iota} a$ is of type A, and A and B are not necessarily convertible.

$$\frac{\Gamma \vdash V : s \quad \Gamma \vdash t : T \quad s \in \{\star, \Box\} \quad x \in \mathcal{X}^s \backslash \mathrm{dom}(\Gamma)}{\Gamma, [x :^a V] \vdash t : T}\ [\textsc{Weak}] \qquad \frac{x \in \mathrm{dom}(\Gamma) \quad \Gamma \vdash x\Gamma : s_x}{\Gamma \vdash x : x\Gamma}\ [\textsc{Var}]$$

$$\frac{}{\vdash \doteq : \forall(T :^u \star).\, T \to T \to \star}\ [\dot{=}\text{-Intro}]$$

$$\frac{\Gamma \vdash T : s_T \quad \Gamma, [x :^a T] \vdash U : s_U}{\Gamma \vdash \forall(x :^a T).\, U : s_U}\ [\textsc{Product}]$$

$$\frac{\Gamma \vdash \forall(x :^a T).\, U : s \quad \Gamma, [x :^a T] \vdash u : U}{\Gamma \vdash \lambda[x :^a T]u : \forall(x :^a T).\, U}\ [\textsc{Lamda}]$$

$$\frac{\begin{array}{c}\Gamma \vdash t : \forall(x :^a U).\, V \quad \Gamma \vdash u : U \\ \text{if } a = r \text{ and } U \xrightarrow{\beta}_* t_1 \doteq_T t_2 \text{ with } t_1, t_2 \in \mathcal{O} \\ \text{then } t_1 \sim_\Gamma t_2 \text{ must hold}\end{array}}{\Gamma \vdash t\,u : V\{x \mapsto u\}}\ [\textsc{App}]$$

$$\frac{}{\vdash \star : \Box}\ [\textsc{Axiom-1}] \qquad \frac{}{\vdash \mathbf{nat} : \star}\ [\textsc{Nat}] \qquad \frac{}{\vdash \mathbf{0} : \mathbf{nat}}\ [\mathbf{0}\text{-Intro}]$$

$$\frac{}{\vdash \mathbf{S} : \mathbf{nat} \to \mathbf{nat}}\ [\mathbf{S}\text{-Intro}] \qquad \frac{}{\vdash \dot{+} : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}}\ [\dot{+}\text{-Intro}]$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash \mathrm{Eq}_T(t) : t \doteq_T t}\ [\textsc{Eq-Intro}]$$

$$\frac{\begin{array}{c}\vdash \tau_{\mathrm{Leib}} : s \\ \tau_{\mathrm{Leib}} = \forall(T : \star)(t_1\, t_2 : T).\, t_1 \doteq t_2 \to \forall(p : T \to \star).\, p\, t_1 \to p\, t_2\end{array}}{\vdash \mathrm{Leib} : \tau_{\mathrm{Leib}}}\ [\textsc{Leib}]$$

$$\frac{\begin{array}{c}\Gamma \vdash t : \mathbf{nat} \quad \Gamma \vdash Q : \mathbf{nat} \to \star \quad \Gamma \vdash f_0 : \mathbf{nat} \\ \Gamma \vdash f_S : \forall(n :^u \mathbf{nat}).\, Q\, n \to Q\, (\mathbf{S}\, n)\end{array}}{\Gamma \vdash \mathrm{Rec}^{\mathcal{W}}_{\mathbb{N}}(t, Q)\{f_0, f_S\} : Q\, t}\ [\iota\text{-Elim}]$$

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s' \quad T \sim_\Gamma T'}{\Gamma \vdash t : T'}\ [\textsc{Conv}]$$

Figure 2.4: Typing judgement of $\mathrm{CC}_{\mathbb{N}}$

In the full version of the calculus, the introduction of strong ι-reduction (i.e. the possibility to construct predicates/types by induction) will reintroduce the problem of type level conversion by allowing conversions at object level to be lifted up at type level. $\mathrm{Rec}^{\mathcal{S}}_{\mathbb{N}}$ denoting here the strong recursor over natural numbers, this is the case for the term

$$V = \mathrm{Rec}^{\mathcal{S}}_{\mathbb{N}}(p, Q)\{\mathbf{nat}, \lambda[n :^u \mathbf{nat}][T :^u Q\, n].\, \mathbf{nat} \to \mathbf{nat}\}$$

Indeed, assuming that $p \sim_\Gamma \mathbf{0}$ and $p \sim_\Gamma \mathbf{1}$, then $V \sim_\Gamma \overset{\iota}{\to} \mathbf{nat}$ and $V \sim_\Gamma \overset{\iota}{\to} \mathbf{nat} \to \mathbf{nat}$.

This problem will be addressed later in the definition of the Calculus of Inductive Congruent Constructions.

Annotations

Annotations are used to ensure that extractable equations are stable by reduction. Without annotations, the property does not hold. For example, the typing of q in $(\lambda[x : t \doteq u]q)\,p$ would make use of the equation $t \doteq u$, whereas the typing of its reduce $q\{x \mapsto p\}$ can no more use the equation (unless $t \doteq u$ is redundant): it has been in some sense *inlined*.

Forbidding the application $(\lambda[x :^r t \doteq u]q)\,p$ (i.e. when the λ-abstraction is annotated with the restricted annotation) fix this problem as it disallows the formation of the β-redex.

One may wonder how annotations can be handled in practice. As seen, annotations are used to forbid *inlining* (when a β-redex is contracted) of equational assumptions used by conversion. This restriction can be removed by using the notion of *opaque definitions* (as opposed to *transparent definitions*) of Coq which allows the user to define symbols that the system cannot inline. In most cases, definitions having a computational behavior (like $\dot{+}$) are transparent whereas definitions representing lemmas (like the associativity of $\dot{+}$) are opaque. This convention is used in the standard library of Coq.

Returning to our previous example, if the user needs to prove a lemma of the form $\forall(x :^r t \doteq u).T$, she should declare it as an opaque definition $P := \lambda[x :^r t \doteq u]q$. The application of P to a term v should then be allowed: the term $P\,v$ cannot reduce to $p\{x \mapsto v\}$. Of course, if P is defined transparently, the application $P\,v$ has to be forbidden as previously.

Moreover, this gives a simple heuristic to automatically tag products and abstractions: the restricted annotation should be used by default when the user is defining an opaque symbol, whereas the unrestricted annotation should be used everywhere else.

CHAPTER 3

The Calculus of Congruent Inductive Constructions

The Calculus of Congruent Inductive Constructions is a modification of the Calculus of Inductive Constructions which embeds in its conversion the validity entailment of a fixed first-order theory over equality.

This chapter is organized as follow. We first recall the full definition of the Calculus of Inductive Construction as described in [48], with the restrictions given in [5]. We then introduce parametric multi-sorted theories. These theories play a crucial role as they will be embeded into the conversion relation of CCIC. Last, we define our calculus, and describe in details how theories are built in the conversion relation.

3.1 The Calculus of Inductive Constructions

Terms of the calculus

We start our presentation by first describing the terms algebra of the Calculus of Inductive Constructions.

CIC uses two *sorts*: $\star$ (or Prop, or *object level universe*) and $\Box$ (or Type, or *predicate level universe*). We denote $\{\star, \Box\}$, the set of CIC sorts, by $\mathcal{S}$.

We use two classes of variables: let $\mathcal{X}^\star$ (resp. $\mathcal{X}^\Box$) be a countably infinite set of *term variables* (resp. *predicate variables*) such that $\mathcal{X}^\star$ and $\mathcal{X}^\Box$ are disjoint. We write $\mathcal{X}$ for $\mathcal{X}^\star \cup \mathcal{X}^\Box$.

We use the following notations:

s	range over	$\mathcal{S}$
$x, y, \ldots$	–	$\mathcal{X}$
$X, Y, \ldots$	–	$\mathcal{X}^\Box$

We can now define the algebra of CIC terms:

Definition 3.1 **Pseudo-terms**

The algebra CIC of pseudo-terms *of CIC is defined as:*

$$t, u, T, U, \ldots := s \in \mathcal{S} \mid x \in \mathcal{X} \mid \forall(x : T).\, t \mid \lambda[x : T].\, t \mid t\, u \mid \mathrm{Ind}(X : t)\{\vec{T_i}\} \mid t^{[n]} \mid \mathrm{Elim}(t : T\,[\vec{u_i}] \to U)\{\vec{w_j}\}$$

Note 4

The notion of free variables is as usual - the binders being λ, $\forall$ and Ind (in $\mathrm{Ind}(X : t)\{\vec{T_i}\}$, X is bound in the T_i's). If $t \in \mathrm{CIC}$, we write $\mathrm{FV}(t)$ for the set of free variables of t. We say that t is closed if $\mathrm{FV}(t) = \varnothing$. A variable x *occurs freely* in t if $x \in \mathrm{FV}(t)$.

Inductive types

The novelty of CIC was to introduce inductive types, denoted by $I = \mathrm{Ind}(X : T)\{\overrightarrow{C_i(X)}\}$ where the $\overrightarrow{C_i(X)}$'s describe the types of the *constructors* of I, and T the type (or *arity*) of I itself, which must be of the form $\forall(\overrightarrow{x_i : T_i}).\star$. The k-th constructor of the inductive type I, of type $C_k\{X \mapsto I\}$, is denoted by $I^{[k]}$.

Example 3.2

The type of natural numbers can be represented by the inductive type $\mathbf{nat} = \mathrm{Ind}(X : \star)\{X, X \to X\}$ where the first constructor $\mathbf{0} = \mathbf{nat}^{[1]}$ (of type $X\{X \mapsto \mathbf{nat}\}$) represents zero, and the second constructor $\mathbf{S} = \mathbf{nat}^{[2]}$ (of type $(X \to X)\{X \mapsto \mathbf{nat}\}$) represents the successor function.

For the consistency of CIC not all inductive types are accepted, but only the ones which are *strictly positive*, a notion that we defined now:

Definition 3.3 **Strictly Positive Inductives Types**

A term $C(X)$ *is a* constructor type in X *if* $C(X) = \forall(\overrightarrow{x_i : U_i}).X\,\vec{u}$ *with* X *not free in* $\vec{u}$. *It is moreover* strictly positive *if for any* i, X *does not occur in* U_i *or* $U_i = \forall(\overrightarrow{z : V}).X\,\vec{v}$ *with* X *not free in* $\vec{V}$ *nor in* $\vec{v}$.

An inductive type $I = \mathrm{Ind}(X : T)\{\overrightarrow{C_i(X)}\}$ *is* strictly positive *if all its constructors are* strictly positive constructor types in X.

Example 3.4

The constructors of $\mathbf{nat} := \mathrm{Ind}(X : \star)\{X, X \to X\}$ are clearly all strictly positive. A more complicated example using the definition of constructor types in all its generality is the one of enumerable ordinals

$$\mathbf{ord} := \mathrm{Ind}(X : \star)\{X, X \to X, (\mathbf{nat} \to X) \to X\}$$

whose constructor types are all strictly positive. See example 3.7 for an example of a non-strictly positive constructor type.

Definition by induction

A definition by induction on an inductive type I is denoted by $\mathrm{Elim}(t : I\,[\overrightarrow{u_i}] \to Q)\{\overrightarrow{v_j}\}$ where the $\overrightarrow{u_i}$'s are the arguments of I, t the term being deconstructed (of type $I\,\overrightarrow{u_i}$), and $Q\,\overrightarrow{u_i}\,t$ the type of the result. The $\overrightarrow{v_j}$'s represent the *branches* of the inductive definition, as explained later.

The reduction relation associated with an inductive definition is the ι-reduction, written $\xrightarrow{\iota}$. For instance, the addition over **nat** can be defined by induction on its first argument as follow:

$$\lambda[x\,y : \mathbf{nat}].\,\mathrm{Elim}(x : \mathbf{nat}\,[\epsilon] \to Q)\{y, \lambda[p\,r : \mathbf{nat}].\,\mathbf{S}\,r\}$$

with $Q = \lambda[\nu : \mathbf{nat}].\,\mathbf{nat}$, given that:

$$\mathrm{Elim}(\mathbf{0} : \mathbf{nat}\,[\epsilon] \to Q)\{\nu_0, \nu_S\} \xrightarrow{\iota} \mathbf{0}$$
$$\mathrm{Elim}(\mathbf{S}\,t : \mathbf{nat}\,[\epsilon] \to Q)\{\nu_0, \nu_S\} \xrightarrow{\iota} \nu_S\,t\,(\mathrm{Elim}(t : \mathbf{nat}\,[\epsilon] \to I)\{\nu_0, \nu_S\}).$$

We now give the general definition of ι-reduction:

Definition 3.5 — **ι-reduction**

Given an inductive definition I*, the* ι*-reduction* $\xrightarrow{\iota}$ *is the smallest rewriting relation such that:*

$$\mathrm{Elim}(T^{[k]}\ \vec{z} : I\,[\vec{u}] \to Q)\{\vec{f}\} \xrightarrow{\iota} \Delta[I, X, C_k, f_k, Q, \vec{f}, \vec{z}]$$

where $I = \mathrm{Ind}(X : \forall(\overrightarrow{x : A}).\star)\{\overrightarrow{C(X)}\}$*, and* $\Delta[I, X, C_k, f_k, Q, \vec{f}, \vec{z}]$ *is defined as follow:*

- $\Delta[I, X, X\,\vec{m}, f, Q, \vec{f}, \epsilon] = f$,
- $\Delta[I, X, \forall(x : B).\,D, f, Q, \vec{f}, z\,\vec{z}] = \Delta[I, X, D\{x \mapsto z\}, f\,z, Q, \vec{f}, \vec{z}]$
  if X *does not occur in* B*,*
- $\Delta[I, X, \forall(x : B).\,D, f, Q, \vec{f}, z\,\vec{z}] =$
  $\Delta[I, X, D\{x \mapsto z\}, f\,z\,(\lambda[\overrightarrow{y : D}].\,\mathrm{Elim}(z\,\vec{y} : I\,[\vec{q}] \to Q)\{\vec{f}\}), Q, \vec{f}, \vec{z}]$
  if $B = \forall(\overrightarrow{y : D}).\,X\,\vec{q}$.

Example 3.6

Although now classical, this definition is quite technical. To illustrate it, we define an inductive predicate **even** of type **nat** → ⋆ s.t. **even** n is inhabited only if n is even (i.e. only if $n = \mathbf{0}$ or $n = \mathbf{S}\,(\mathbf{S}\,k)$ with **even** k inhabited):

$$\mathbf{even} = \mathrm{Ind}(X : \mathbf{nat} \to \star)\{X\,\mathbf{0}, \forall(n : \mathbf{nat}).\,X\,n \to X\,(\mathbf{S}\,(\mathbf{S}\,n))\}$$

We write **EvenO** for $\mathbf{even}^{[1]}$, and **EvenS** for $\mathbf{even}^{[2]}$. We now define a function div2 which takes a natural number $k : \mathbf{nat}$, a proof $p : \mathbf{even}\,k$ and computes k/2:

$$\mathrm{div2} = \lambda[k : \mathbf{nat}][p : \mathbf{even}\,k].\,\mathrm{Elim}(p : \mathbf{even}\,[k] \to Q)\{f_1, f_2\}$$

where $f_1 = \mathbf{0}$ and $f_2 = \lambda[k' : \mathbf{nat}][p' : \mathbf{even}\,k'][r : \mathbf{nat}].\,\mathbf{S}\,r$.

We do not explicit the form of Q which is only used for typing purposes.

Given that a closed and $\xrightarrow{\beta\iota}$-normal term p of type $\mathbf{even}(\overbrace{\mathbf{S}\cdots(\mathbf{S}}^{2*k}\ \mathbf{0}))$ must be of the form $(\overbrace{\mathbf{EvenS}\cdots(\mathbf{EvenS}}^{k}\ \mathbf{0}\,\mathbf{EvenO}))$ (this is a consequence of the *inversion lemma* for CIC), div2 simply proceeds by "counting the occurrences of **EvenS** in p". We show in Figure 3.1 how $\mathrm{div2}\ (\mathbf{S}\,(\mathbf{S}\,\mathbf{0}))\ (\mathbf{EvenS}\,\mathbf{0}\,\mathbf{EvenO})$ reduces to $\mathbf{S}\,\mathbf{0}$.

Example 3.7 — **Non positive constructor type**

We here give a non positive type, where the occurrence of X which does not appear in strict positive position is underlined:

$$\mathbf{absurd} := \mathrm{Ind}(X : \star)\{(\underline{X} \to X) \to X\}$$

If such a definition were allowed, it would be possible to construct non-strongly normalizing terms. For example, if f denotes the term:

$$f := \lambda[x : \mathbf{absurd}].\,\mathrm{Elim}(x : \mathbf{absurd}\,[\epsilon] \to \lambda[y : \mathbf{absurd}].\,\mathbf{absurd})\{b\}$$

with $b := \lambda[v : \mathbf{absurd} \to \mathbf{absurd}].\,v\,(\mathbf{absurd}^{[1]}\,v)$, then one can verify that

$$f\,(\mathbf{absurd}^{[1]}\,f) \xrightarrow{\beta\iota}_{*} f\,(\mathbf{absurd}^{[1]}\,f).$$

$$
\begin{aligned}
& \mathrm{div2}\,(\mathbf{S}\,(\mathbf{S}\,\mathbf{0}))\,(\mathbf{EvenS}\,\mathbf{0}\,\mathbf{EvenO}) \\
\xrightarrow{\beta}_{+}\ & \mathrm{Elim}(\mathbf{EvenS}\,\mathbf{0}\,\mathbf{EvenO} : \mathbf{even}\,[\mathbf{S}\,(\mathbf{S}\,\mathbf{0})] \to Q)\{f_1, f_2\} \\
\xrightarrow{\iota}\ & \Delta[\mathbf{even}, X, \forall(n : \mathbf{nat}).\, X\,n \to X\,(\mathbf{S}\,(\mathbf{S}\,n)), f_2, Q, \vec{f_i}, [\mathbf{0}\,\mathbf{EvenO}]] \\
=\ & \Delta[\mathbf{even}, X, X\,\mathbf{0} \to X\,(\mathbf{S}\,(\mathbf{S}\,\mathbf{0})), f_2\,\mathbf{0}, Q, \vec{f_i}, [\mathbf{EvenO}]] \\
=\ & \Delta[\mathbf{even}, X, X\,(\mathbf{S}\,(\mathbf{S}\,\mathbf{0})), f_2\,\mathbf{0}\ \mathrm{Elim}(\mathbf{EvenO} : \mathbf{even}\,[\mathbf{0}] \to Q)\{f_1, f_2\}, Q, \vec{f_i}, \epsilon] \\
=\ & f_2\,\mathbf{0}\,\mathbf{EvenO}\ \mathrm{Elim}(\mathbf{EvenO} : \mathbf{even}\,[\mathbf{0}] \to Q)\{f_1, f_2\} \\
\xrightarrow{\iota}\ & f_2\,\mathbf{0}\,\mathbf{EvenO}\,\Delta[\mathbf{even}, X, X\,\mathbf{0}, f_1, Q, \vec{f_i}, []] \\
=\ & f_2\,\mathbf{0}\,\mathbf{EvenO}\,f_1 \xrightarrow{\beta}_{+} \mathbf{S}\,\mathbf{0}
\end{aligned}
$$

Figure 3.1: CIC ι-reduction example

Strong and Weak ι-reduction

CIC distinguishes two kinds of ι-elimination: the *strong* one, when the terms constructed by induction are at predicate level, and the *weak* one, when they are at object level. (The categorization of terms which defines these levels is given later) To ensure logical consistency, strong ι-elimination is restricted to *small* inductive types, i.e. to inductive types whose constructors do not take a predicate as argument:

Definition 3.8 **Small inductive types**

A type constructor $\forall(\overrightarrow{x_i : T_i}).\, X\,\vec{t}$ *in* X *is* small *if all the* x_i*'s are in* $\mathcal{X}^\star$ *(or equivalently for terms that are typable, if all the* T_i*'s are of type* $\star$ *in their respective environments). If not, it is called a* big *type constructor. An inductive type is* small *if all its constructor types are small.*

Example 3.9

The inductive type $I := \mathrm{Ind}(X : \star)\{\star \to X\}$ is not small. Allowing strong reduction on such an inductive definition, we can define two terms roll and unroll (of types $I \to \star$ and $\star \to I$) s.t. $\mathrm{unroll}\,(\mathrm{roll}\,x) \xrightarrow{\beta\iota}_{*} x$:

$$
\begin{aligned}
\mathrm{roll} &= I^{[1]} \\
\mathrm{unroll} &= \lambda[x : I].\, \mathrm{Elim}(x : I\,[\epsilon] \to Q)\{[\lambda[v : \star].\, v]\}
\end{aligned}
$$

Having such terms allows the encoding in CIC of a typed version of the Burali-Forti paradox [11].

Typing judgments

The typing judgments are classically written $\Gamma \vdash t : T$, meaning that the *well-formed term* t is a proof of the proposition T (or has type T) under the *well-formed environment* Γ, where environments are defined as follows:

Definition 3.10 **Typing environments of CIC**

A typing environment Γ *is a sequence of pairs* $[x_i : T_i]$ *made of a variable* x_i *and a term* T_i *(we say that* Γ *binds* x_i *to the type* T_i*), such that* Γ *does not bind a variable twice.* Γ *can be seen as a substitution :* $x\Gamma$ *will denote the type associated to* x *in* Γ*, and we write*

$\mathrm{dom}(\Gamma)$ *for the domain of* Γ *as well.*

Before describing how typing judgments are formed, we first define the notion of *constructor derivation*. Constructor derivations are used for typing the branches of an inductive definition.

Definition 3.11 **Constructor derivation**

Given an inductive definition I*, we define:*

- *the* $\star$*-level constructor derivation* $\Delta^\star\{\mathsf{I}, \mathsf{X}, \mathsf{C}, \mathsf{Q}, \mathsf{c}\}$ *as:*
  - $\Delta^\star\{\mathsf{I}, \mathsf{X}, \mathsf{X}\,\vec{\mathsf{m}}, \mathsf{Q}, \mathsf{c}\} = \mathsf{Q}\,\vec{\mathsf{m}}\,\mathsf{c}$
  - $\Delta^\star\{\mathsf{I}, \mathsf{X}, \forall(z : \mathsf{B}).\,\mathsf{D}, \mathsf{Q}, \mathsf{c}\} = \forall(z : \mathsf{B}).\,\Delta\{\mathsf{I}, \mathsf{X}, \mathsf{D}, \mathsf{Q}, \mathsf{c}\,z\}$ *if* X *does not occur in* B
  - $\Delta^\star\{\mathsf{I}, \mathsf{X}, \forall(z : \mathsf{B}).\,\mathsf{D}, \mathsf{Q}, \mathsf{c}\} =$
    $\forall(x : \mathsf{B}\{\mathsf{X} \mapsto \mathsf{I}\}).\,(\forall(\overrightarrow{y : \mathsf{D}}).\,\mathsf{Q}\,\vec{q}\,(z\,\vec{y})) \to \Delta\{\mathsf{I}, \mathsf{X}, \mathsf{D}, \mathsf{Q}, \mathsf{c}\,z\}$
    if $\mathsf{B} = \forall(\overrightarrow{y : \mathsf{D}}).\,\mathsf{X}\,\vec{q}$
- *the* $\Box$*-level constructor derivation* $\Delta^\Box\{\mathsf{I}, \mathsf{X}, \mathsf{C}, \vec{x}\,y, \mathsf{K}, \mathsf{c}\}$ *as:*
  - $\Delta^\Box\{\mathsf{I}, \mathsf{X}, \mathsf{X}\,\vec{\mathsf{m}}, \vec{x}\,y, \mathsf{K}, \mathsf{c}\} = \mathsf{K}\{\overrightarrow{x \mapsto \mathsf{m}}, y \mapsto \mathsf{c}\}$
  - $\Delta^\Box\{\mathsf{I}, \mathsf{X}, \forall(z : \mathsf{B}).\,\mathsf{D}, \vec{x}\,y, \mathsf{K}, \mathsf{c}\} = \forall(z : \mathsf{B}).\,\Delta^\Box\{\mathsf{I}, \mathsf{X}, \mathsf{D}, \vec{x}\,y, \mathsf{K}, \mathsf{c}\,z\}$
    if X *does not occur in* B
  - $\Delta^\Box\{\mathsf{I}, \mathsf{X}, \forall(z : \mathsf{B}).\,\mathsf{D}, \vec{x}\,y, \mathsf{K}, \mathsf{c}\} =$
    $\forall(z : \mathsf{B}\{\mathsf{X} \mapsto \mathsf{I}\}).\,(\forall(\overrightarrow{y : \mathsf{D}}).\,\mathsf{K}\{\overrightarrow{x \mapsto \mathsf{m}}, y \mapsto \mathsf{c}\}) \to \Delta^\Box\{\mathsf{I}, \mathsf{X}, \mathsf{D}, \vec{x}\,y, \mathsf{K}, \mathsf{c}\,z\}$
    if $\mathsf{B} = \forall(\overrightarrow{y : \mathsf{D}}).\,\mathsf{X}\,\vec{q}$

We can now define the formation of $\Gamma \vdash \mathsf{t} : \mathsf{T}$:

Definition 3.12 **Typing judgment of CIC**

Typing rules of CIC are defined in Figure 3.2 and 3.3.

3.2 Parametric multi-sorted theories with constructors

We choose to embed into the Calculus of Constructions any first-order theory expressed by a parametric multi-sorted algebra, with some restrictions for the notion of constructor symbols. These algebras can be easily mapped to the Calculus of Inductive Construction and are expressive enough to describe any theory we want to embed in the calculus and for which a decision procedure exists: linear arithmetic, datatypes, non-interpreted algebras, rings.

Signature

The first part of this section is taken from the definition of parametric multi-sorted algebras, with some restrictions for the introduction of constructor symbols.

Definition 3.13

A set of sort constructors *is any finite set* Λ *whose elements are equipped with an arity. If*

$$\frac{}{\vdash \star : \square}\ [\textsc{Ax-1}]$$

$$\frac{\Gamma \vdash T : s_T \quad \Gamma, [x : T] \vdash U : s_U}{\Gamma \vdash \forall (x : T).\, U : s_U}\ [\textsc{Prod}]$$

$$\frac{\Gamma \vdash \forall (x : T).\, U : s \quad \Gamma, [x : T] \vdash u : U}{\Gamma \vdash \lambda [x : T].\, u : \forall (x : T).\, U}\ [\textsc{Lam}]$$

$$\frac{\Gamma \vdash V : s \quad \Gamma \vdash t : T \quad s \in \{\star, \square\} \quad x \in \mathcal{X}^s \setminus \mathrm{dom}(\Gamma)}{\Gamma, [x : V] \vdash t : T}\ [\textsc{Weak}]$$

$$\frac{x \in \mathrm{dom}(\Gamma) \cap \mathcal{X}^{s_x} \quad \Gamma \vdash x\Gamma : s_x}{\Gamma \vdash x : x\Gamma}\ [\textsc{Var}]$$

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash T : s \quad \Gamma \vdash T' : s' \quad T \overset{\beta\iota}{\longleftrightarrow}_* T'}{\Gamma \vdash t : T'}\ [\textsc{Conv}]$$

Figure 3.2: CIC Typing Rules (CC rules)

$$\frac{\begin{array}{c} A = \forall (\overrightarrow{x : T}).\, \star \quad \vdash A : \square \quad \text{for all } i,\ \Gamma \vdash C_i(X) : \star \\ \text{for all } i,\ C_i(X) \text{ is a strictly positive constructor in } X \\ I = \mathrm{Ind}(X : A)\{\overrightarrow{C_i(X)}\} \text{ is in } \overset{\beta\iota}{\longrightarrow}\text{-normal form} \end{array}}{\Gamma \vdash I : A}\ [\textsc{Ind}]$$

$$\frac{I = \mathrm{Ind}(X : T)\{\overrightarrow{C_i(X)}\} \quad \Gamma \vdash I : T}{\Gamma \vdash I^{[k]} : C_k\{X \mapsto I\}}\ [\textsc{Constr}]$$

$$\frac{\begin{array}{c} A = \forall (\overrightarrow{x : U}).\, \star \quad I = \mathrm{Ind}(X : A)\{\overrightarrow{C_j(X)}\} \quad \Gamma \vdash I : A \quad \vdash Q : \forall (\overrightarrow{x : U}).\, (I\ \vec{x}) \to \star \\ \text{for all } i,\ T_i = \Delta^{\star}\{I, X, C_i(X), Q, I^{[i]}\} \quad \vdash T_i : \star \quad \Gamma \vdash f_i : T_i \\ \text{for all } j,\ \Gamma \vdash a_j : A_j\{\overrightarrow{x \mapsto a}\} \quad \Gamma \vdash c : I\ \vec{a} \end{array}}{\Gamma \vdash \mathrm{Elim}(c : I\,[\vec{a}] \to Q)\{\vec{f}\} : Q\ \vec{a}\ c}\ [\textsc{Elim-}\star]$$

$$\frac{\begin{array}{c} A = \forall (\overrightarrow{x : U}).\, \star \quad I = \mathrm{Ind}(X : A)\{\overrightarrow{C_j(X)}\} \text{ is small} \\ Q = \forall (\overrightarrow{x : U})(y : I\ \vec{x}).\, K \text{ is in } \overset{\beta\iota}{\longrightarrow}\text{-normal form} \quad [\overrightarrow{x : U}], [y : I\ \vec{x}] \vdash K : \square \\ \text{for all } i,\ T_i = \Delta^{\square}\{I, X, C_i(X), \vec{x}\ y, K, I^{[i]}\} \quad \vdash T_i : \square \quad \Gamma \vdash f_i : T_i \\ \text{for all } j,\ \Gamma \vdash a_j : A_j\{\overrightarrow{x \mapsto a}\} \quad \Gamma \vdash c : I\ \vec{a} \end{array}}{\Gamma \vdash \mathrm{Elim}(c : I\,[\vec{a}] \to Q)\{\vec{f}\} : K\{\overrightarrow{x \mapsto}, y \mapsto c\}}\ [\textsc{Elim-}\square]$$

Figure 3.3: CIC Typing Rules (Inductive Types)

$\mathcal{E}$ is a countably infinite set disjoint from Λ of sort variables*, the free algebra $\Lambda_{\mathcal{E}} = \mathcal{T}(\Lambda, \mathcal{E})$ is called the* set of (first-order) parametric sorts*.*

Notation. From now on,

$\alpha, \beta, \ldots$	ranges over	$\mathcal{E}$
$\underline{\sigma}, \underline{\tau}, \ldots$	-	Λ
$\sigma, \tau, \ldots$	-	$\Lambda_{\mathcal{E}}$

If $\underline{\sigma}$ is a sort constructor, we write $\underline{\sigma}/n$ for meaning that $\underline{\sigma}$ if of arity n.

Example 3.14

Examples of sorts are the sort **nat** of natural numbers, or **list**(**nat**) of lists over natural numbers, **nat** (resp. **list**) being a sort constructor of arity 0 (resp. of arity 1).

Definition 3.15

A signature *in* parametric multi-sorted algebra *is a pair* $(\Lambda_{\mathcal{E}}, \Sigma)$ *where i)* $\Lambda_{\mathcal{E}}$ *is a set of sorts, ii)* Σ *is a finite set, disjoint from all others, of* function symbols*, and iii) an arity of the form* $\forall \vec{\alpha}. \tau_1 \times \cdots \tau_n \to \sigma$ *is attached to any symbol* $f \in \Sigma$*, where* $\vec{\alpha}$ *if the set of sort variables occurring in* $\tau_1, \ldots, \tau_n, \sigma$ *and all the sort variables occurring in* σ *occur in the* τ_i*'s.*

We distinguish a subset Σ^c *of* Σ*, called set of* constructor symbols*, the arity of which must be of the form* $\forall \alpha_1, \ldots, \alpha_n. \tau_1 \times \cdots \times \tau_n \to \underline{\sigma}(\alpha_1, \ldots, \alpha_n)$.

Function symbols of $\Sigma - \Sigma^c$ *are called* defined symbols.

Example 3.16

Returning to our previous example, the signature of parametric lists is given by $\Lambda = \{\mathbf{list}/1\}$ and the signature:

nil	$\forall \alpha.$	$\to \mathbf{list}(\alpha)$
cons	$\forall \alpha. \alpha \times \mathbf{list}(\alpha)$	$\to \mathbf{list}(\alpha)$
car	$\forall \alpha. \mathbf{list}(\alpha)$	$\to \alpha$
cdr	$\forall \alpha. \mathbf{list}(\alpha)$	$\to \mathbf{list}(\alpha)$

whereas the one of natural numbers is given by $\Lambda = \{\mathbf{nat}/0\}$ and:

0		$\to \mathbf{nat}$
S	**nat**	$\to \mathbf{nat}$
$\dot{+}$	**nat** $\times$ **nat**	$\to \mathbf{nat}$

Terms, Equations

We continue with the definition of terms and equations:

Definition 3.17 **Terms**

Let (Λ, Σ) *be a signature in parametric multi-sorted algebra. For any* $\sigma \in \Lambda_{\mathcal{E}}$*, let* $\mathcal{X}^\sigma$ *be a countable infinite set of* variables of sort σ*, s.t. all the* $\mathcal{X}^\tau$*'s are pairwise disjoint. Let* $\mathcal{X} = \bigcup_{\sigma \in \Lambda_{\mathcal{E}}} \mathcal{X}^\sigma$*. For any* $x \in \mathcal{X}$*, we say that* x *has sort* τ *if* $x \in \mathcal{X}^\tau$.

For any sort $\sigma \in \Lambda$*, we define the set* $\mathcal{T}_\sigma(\Sigma, \mathcal{X})$ *of* terms of sort σ with variables in $\mathcal{X}$ *as the smallest set s.t.:*

1. *if* $x \in \mathcal{X}^\tau$*, then* $x \in \mathcal{T}_\tau(\Sigma, \mathcal{X})$,
2. *a) if* $t_1, \cdots, t_n \in \mathcal{T}_{\sigma_1 \xi}(\Sigma, \mathcal{X}) \times \cdots \times \mathcal{T}_{\sigma_n \xi}(\Sigma, \mathcal{X})$*, for a sort substitution* ξ*,*
   b) $f : \sigma_1 \times \cdots \times \sigma_n \to \tau$*,*

then $f(t_1, \ldots, t_n) \in \mathcal{T}_{\tau\xi}(\Sigma, \mathcal{X})$.

We denote by $\mathcal{T}(\Sigma, \mathcal{X})$ *the set* $\bigcup_{\sigma \in \Lambda_\mathcal{E}} (\mathcal{T}_\sigma(\Sigma, \mathcal{X}))$.

We also introduce the notion of *term schemes*, use later for the definition of theories and rewriting systems.

Definition 3.18 — **Term scheme**

Let $\mathcal{X}^\top$ *be a set of unsorted variables. A sort assignment is any finite mapping from* $\mathcal{X}^\top$ *to* $\Lambda_\mathcal{E}$*, written* $[x_1 : \tau_1], \ldots, [x_n : \tau_n]$*. We write* $\mathcal{I}_\mathcal{S} x$ *for the sort associated to* x *in the sort assignment* $\mathcal{I}_\mathcal{S}$*.*

Given a sort assignment $\mathcal{I}_\mathcal{S} = [x_1 : \tau_1], \ldots, [x_n : \tau_n]$*, we inductively defined the* term schemes of sort σ w.r.t. $\mathcal{I}_\mathcal{S}$ *as:*

1. $\mathcal{I}_\mathcal{S}.x_i$ *is a term scheme of sort* τ_i*,*
2. *a) if* $\mathcal{I}_\mathcal{S}.t_1, \cdots, \mathcal{I}_\mathcal{S}.t_n$ *are term schemes of respective sorts* $\sigma_1\xi, \ldots, \sigma_n\xi$ *for a sort renaming* ξ*,*
   b) $f : \sigma_1 \times \cdots \times \sigma_n \to \tau$*,*

   then $\mathcal{I}_\mathcal{S}.f(t_1, \ldots, t_n)$ *is a term scheme of sort* $\tau\xi$*.*

Let $\bigcup_{\sigma \in \Lambda_\mathcal{E}} \mathcal{X}^\sigma$ *be a set of sorted variables. For any variable* $x \in \mathcal{X}^\top$ *and sort* σ*, we assign to* x *and* σ *a unique variable of* $\mathcal{X}^\sigma$*, denoted by* x^σ*. Given a sort substitution* ξ*, the* ξ-instance of a term scheme $\mathcal{I}_\mathcal{S}.t$*, written* $\langle \mathcal{I}_\mathcal{S}.t \rangle_\xi$ *is inductively defined as:*

1. $\langle x_i \rangle_\xi = x_i^{(x_i \mathcal{I}_\mathcal{S})\xi}$*,*
2. $\langle \mathcal{I}_\mathcal{S}.f(t_1, \ldots, t_n) \rangle_\xi = f(\langle \mathcal{I}_\mathcal{S}.t_1 \rangle_\xi, \ldots, \langle \mathcal{I}_\mathcal{S}.t_n \rangle_\xi)$.

Fact 3.19

If $\mathcal{I}_\mathcal{S}.t$ is a term scheme of sort σ and ξ a sort substitution, then $\langle t \rangle_\xi$ is a term of sort $\sigma\xi$.

Example 3.20

Using the signature of example 3.16, with x a variable of sort **nat**, we have 0, $0 + x$ of sort **nat**, and **nil** of sort **list**(**nat**) and sort **list**(α).

The term scheme $[x : \alpha].\mathbf{cons}(x, \mathbf{nil})$ is of sort **list**(α).

Definition 3.21 — **Substitution**

Let (Λ, Σ) *be a signature and* $\mathcal{X} = \bigcup_{\sigma \in \Lambda_\mathcal{E}} \mathcal{X}^\sigma$ *be a set of variables. A* substitution *is any finite mapping, written* $\{x_i \mapsto t_i\}_{1 \leqslant i \leqslant n}$*, from* $\mathcal{X}$ *to* $\mathcal{T}(\Sigma, \mathcal{X})$ *preserving sorts, that is s.t. for all* $x \in \mathrm{dom}(\theta)$ *of sort* σ*,* $x\theta \in \mathcal{T}_\sigma(\Sigma, \mathcal{X})$*.*

If θ *is an* $\mathcal{T}(\Sigma, \mathcal{X})$*-substitution, we write* $\mathrm{dom}(\theta) = \{x_1, \ldots, x_n\}$ *for the* domain of θ*. If* t *is a term of sort* σ*, we denote by* $t\theta$ *the term where all occurrences of the* x_i*'s has been respectively replaced with the* t_i*'s.*

Fact 3.22

If t is a term of sort σ, then $t\theta$ is of sort σ for any substitution θ.

Equations are simply pairs of terms of same sort:

Definition 3.23 — **Equation**

Let $(\Lambda_{\mathcal{E}}, \Sigma)$ be a signature in parametric multi-sorted algebra. A Σ-equation is any triple (t, u, σ), where t and u are terms of sort σ, written $t =^{\sigma} u$ or $t = u$ when σ is not relevant.

A Σ-equation scheme *is any triple (t, u, σ) where $\mathcal{I}_{\mathcal{S}}.t$ and $\mathcal{I}_{\mathcal{S}}.u$ are* term schemes *of sort σ, written $\forall \overline{\alpha}. \forall \overrightarrow{x_i : \tau_i}. t = u$, where the x_i's (written with their associated sorts) are the variables appearing in t and u and the $\vec{\alpha}$'s are the sort variables appearing in the τ_i's.*

Algebra, Interpretation

We can now give the interpretation of signatures and terms:

Definition 3.24 — **Algebra**

Let $\Omega = (\Lambda_{\mathcal{E}}, \Sigma)$ be a signature in parametric multi-sorted algebra.

A Ω-algebra $\mathcal{A}$ *consists of*

1. *a domain $\mathcal{S}$ for sort constructors interpretation,*
2. *for any sort constructor $\underline{\sigma}$, a function $S_{\underline{\sigma}} : \mathcal{S}^n \to \mathcal{S}$, where n is the arity of $\underline{\sigma}$,*
3. *for each function symbol $f \in \Sigma$ of arity $\forall \vec{\alpha}. \tau_1 \times \cdots \times \tau_n \to \sigma$, a function:*
$$A_f : \mathcal{A}_{\tau_1} \times \cdots \times \mathcal{A}_{\tau_n} \to \mathcal{A}_{\sigma}.$$
where $\mathcal{A}_\nu$, for $\nu \in \mathcal{T}(\Lambda)$, is inductively defined as:
$$\mathcal{A}_{\underline{\nu}(\pi_1, \ldots, \pi_k)} = \mathcal{S}_{\underline{\nu}}(\mathcal{A}_{\pi_1}, \ldots, \mathcal{A}_{\pi_k})$$
and, for $\nu \in \Lambda_{\mathcal{E}} \backslash \mathcal{T}(\Lambda)$,
$$\mathcal{A}_\nu = \bigcup \{\mathcal{A}_{\nu\theta} \mid \theta \text{ closed}\}.$$
Moreover, for any sort substitution ξ, if $(\nu_1, \ldots, \nu_n) \in \mathcal{A}_{\tau_1 \xi} \times \cdots \times \mathcal{A}_{\tau_n \xi}$, then $A_f(\nu_1, \ldots, \nu_n) \in \mathcal{A}_{\tau\xi}$.

Definition 3.25 — **Interpretation**

Let Ω be a signature in parametric multi-sorted algebra, $\mathcal{X} = \bigcup_{\sigma \in \Lambda_{\mathcal{E}}} \mathcal{X}^{\sigma}$ a set of sorted variables, and $\mathcal{A}$ a Ω-algebra. An assignment is any function $\mathcal{I} : \mathcal{X} \to \bigcup_{\sigma \in \Lambda_{\mathcal{E}}} \mathcal{A}_{\sigma}$ respecting sorts, i.e. such that $\mathcal{I}(x) \in \mathcal{A}_\tau$ if $x \in \mathcal{X}^\tau$.

The $\mathcal{A}$-interpretation of $t \in \mathcal{T}(\Sigma, \mathcal{X})$ with assignment $\mathcal{I}$, written $[\![t]\!]^{\mathcal{I}}_{\mathcal{A}}$, is defined as:

$$[\![x]\!]^{\mathcal{I}}_{\mathcal{A}} = \mathcal{I}(x)$$
$$[\![f(t_1, \ldots, t_n)]\!]^{\mathcal{I}}_{\mathcal{A}} = \mathcal{A}_f([\![t_1]\!]^{\mathcal{A}}_{\mathcal{I}}, \ldots, [\![t_n]\!]^{\mathcal{A}}_{\mathcal{I}})$$

Theory, Model

We now move to the definition of theories:

Definition 3.26 — **Theory**

Let Ω be a signature in parametric multi-sorted algebra. A Ω-theory *is any set of equation schemes.*

The definition of a model of a theory is as usual. We take into account here that we have constructor symbols.

Definition 3.27 — **Model**

Let $\mathcal{T}$ be a Ω-theory. A model *of $\mathcal{T}$ (or $\mathcal{T}$-model for short) is any Ω-algebra $\mathcal{M}$ s.t. for any assignment $\mathcal{I}$:*

- *if f is a constructor symbol,*
$$[\![f(t_1,\cdots,t_n)]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![f(u_1,\cdots,u_n)]\!]^{\mathcal{I}}_{\mathcal{M}} \text{ implies for all } i,\ [\![t_i]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![u_i]\!]^{\mathcal{I}}_{\mathcal{M}}$$
- *if f and g are two distinct constructor symbols, then for any $t_1,\cdots,t_n,u_1,\cdots,u_k$:*
$$[\![f(t_1,\cdots,t_n)]\!]^{\mathcal{I}}_{\mathcal{M}} \neq [\![g(u_1,\cdots,u_k)]\!]^{\mathcal{I}}_{\mathcal{M}}$$
- *for any equation scheme $\forall\overrightarrow{\alpha}.\forall\overrightarrow{x_i:\tau_i}.\,t = u$ of the theory, and any sort substitution ξ, we have:*
$$[\![\langle\mathcal{I}_S.\,t\rangle_\xi]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![\langle\mathcal{I}_S.\,u\rangle_\xi]\!]^{\mathcal{I}}_{\mathcal{M}}.$$
with $\mathcal{I}_S = [\overrightarrow{x_i:\tau_i}]$.

Sentence, $\mathcal{T}$-validity

Let $\mathcal{T}$ be a Ω-theory. In all the following, we will only consider *Horn clauses*, i.e. sentences of the form $t_1 =_{\sigma_1} u_1 \wedge\cdots\wedge t_n =_{\sigma_n} u_n \Rightarrow t =_\sigma u$.

Validity of such sentences w.r.t. a $\mathcal{T}$-model is defined in the obvious way:

Definition 3.28 — **Validity**

We say that $\overbrace{t_1 =_{\sigma_1} u_1 \wedge\cdots\wedge t_n =_{\sigma_n} u_n}^{E} \Rightarrow t =_\sigma u$ *is* valid *under a $\mathcal{T}$-model $\mathcal{M}$ and an $\mathcal{M}$-interpretation $\mathcal{I}$, written $\mathcal{M}\models_{\mathcal{I}} E \Rightarrow t =_\sigma u$, if either:*

- *there exists an equation $t_i =_{\sigma_i} u_i$ in E s.t. $[\![t_i]\!]^{\mathcal{I}}_{\mathcal{M}} \neq [\![u_i]\!]^{\mathcal{I}}_{\mathcal{M}}$, or*
- $[\![t]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![u]\!]^{\mathcal{I}}_{\mathcal{M}}$.

We say that $E \Rightarrow t =_\sigma u$ is valid under the model $\mathcal{M}$ if for any $\mathcal{M}$-interpretation $\mathcal{I}$, $E \Rightarrow t =_\sigma u$ is valid under $\mathcal{M}$ and $\mathcal{I}$.

If E is any possibly infinite set of equations, we write $\mathcal{M}, E \models_{\mathcal{I}} t =_\sigma u$ (resp. $\mathcal{M}, E \models t =_\sigma u$) if there exists a finite subset F of E s.t. $\mathcal{M}\models_{\mathcal{I}} \bigwedge F \Rightarrow t =_\sigma u$ (resp. $\mathcal{M}\models \bigwedge F \Rightarrow t =_\sigma u$).

Validity of sentences for a given $\mathcal{T}$-theory is defined as the validity under all the considered $\mathcal{T}$-models. We write $\mathcal{T}, E \models t =_\sigma u$ if $\mathcal{M}, E \models t =_\sigma u$ for all the considered $\mathcal{T}$-models.

Rewriting systems

We end this section by introducing the notion of parametric rewriting systems. A rewrite rule is simply an equation scheme oriented from left to write.

Definition 3.29 — **Rewriting system**

Let $(\Lambda_{\mathcal{E}}, \Sigma)$ *be a signature in parametric multi-sorted algebra. A* Σ-rewrite rule *is any triple* (t, u, σ) *of term* schemes *of sort* σ*, written* $\forall \overline{\alpha}. \forall \overrightarrow{x_i : \tau_i}. t \to u$*, where the* x_i*'s (written with their associated sorts) are the variables appearing in* t *and* u *and the* $\vec{\alpha}$*'s are the sort variables appearing in the* τ_i*'s.*

A Σ rewriting system *is any enumerable set of* Σ*-rewrite rules.*

Rewriting of terms is defined as usual:

Definition 3.30

Let R *be a* Σ*-rewrite system. The rewriting relation* $\xrightarrow{R}$ *associated to* R *if the smallest binary relation on* Σ*-terms, closed by substitution and context, s.t. for any rewrite rule* $\forall \overline{\alpha}. \forall \overrightarrow{x_i : \tau_i}. t \to u$ *of* R *and any sort substitution* $\xi \in \Lambda_{\mathcal{E}}$*, we have:*

$$\langle \mathcal{I}_S . t \rangle_\xi \xrightarrow{R} \langle \mathcal{I}_S . u \rangle_\xi$$

with $\mathcal{I}_S = [\overrightarrow{x_i : \tau_i}]$.

Fact 3.31

The relation $\xrightarrow{R}$ preserves term sorts.

3.3 The calculus

Terms of the calculus

As for CIC, we first start by describing the terms algebra of the calculus.

CCIC uses the same set of sorts $\mathcal{S} = \{\star, \square\}$ and sets of variables $\mathcal{X} = \mathcal{X}^\star \cup \mathcal{X}^\square$ of CIC. We distinguish a subset $\mathcal{X}^-_\star$ of $\mathcal{X}^\star$ of *extractable variables* (resp. $\mathcal{X}^-_\square$ of *extractable types variables*).

As for $CC_{\mathbb{N}}$, let $\mathcal{A} = \{r, u\}$ a set of two constants, called *annotations* where r stands for *restricted* and u for *unrestricted*. We use a for an arbitrary annotation.

Let $\underline{\Sigma}$ and $\underline{\Lambda}$ be two disjoint sets of function symbols. We define $\underline{\Sigma}$ and $\underline{\Lambda}$ later, when translating parametric multi-sorted signature to CCIC terms: $\underline{\Sigma}$ (resp. $\underline{\Lambda}$) will then contain the translation of the first-order function symbols (resp. of the sort constructors).

Definition 3.32 — **Pseudo-terms of CCIC**

The algebra CCIC of pseudo-terms *of CCIC is defined by:*

$$\begin{aligned} t, u, T, U, \ldots := {} & s \in \mathcal{S} \mid x \in \mathcal{X} \mid \forall (x :^a T).\, t \mid \lambda [x :^a T].\, t \mid t\, u \mid f \in \underline{\Sigma} \mid \sigma \in \underline{\Lambda} \\ & \mid \doteq \mid \mathrm{Eq}_T(t) \mid \mathrm{Leib} \mid \mathrm{Ind}(X : t)\{\overrightarrow{T_i}\} \mid t^{[n]} \mid \mathrm{Elim}(t : T\, [\overrightarrow{u_i}] \to U)\{\overrightarrow{w_j}\} \end{aligned}$$

We assume that for any $f \in \underline{\Sigma} \cup \underline{\Lambda}$*, a CCIC term* τ_f*, called the* type of f*, is attached to* f.

Compared to CIC, the differences are:

· the presence of annotations for the product and abstraction,

- the internalization of the equality predicate:
  1. $t \doteq_T u$ (or $t \doteq u$ when T is not relevant or can deduced from the context) denotes the equality of the two terms (of type T) t and u,
  2. $\mathrm{Eq}_T(t)$ will represent proof by reflexivity of $t \doteq_T t$ and Leib the Leibniz predicate for equality,
- the internalization of the function symbols $f \in \Sigma$ and sort constructors $\sigma \in \Lambda$.

Notation. When x in not free in t, $\forall(x :^a T).t$ will be written $T \to^a t$. The default annotation, when not specified in a product or abstraction, is the *unrestricted* (u) one.

From now on, let $\mathcal{O}^+$ and $\mathcal{O}^-$ be two arbitrary set of CCIC terms, whose elements are called *extractable terms* and *convertible terms*. The set $\mathcal{O}^+$ will be used in the definition of conversion to restrict the set of *extractable* equations for a given environment: only equation of the form $t \doteq u$ with t and u in $\mathcal{O}^+$ shall be extractable. The set $\mathcal{O}^-$ is used later to restrict the set of terms on which first-order deduction is applied.

Taking $\mathcal{O}^+ = \mathcal{O}^- = \mathcal{O}$ (see Definition 3.34) does not compromise most standard calculus properties but it does compromise decidability of type-checking. For example, if $\mathcal{T}$ is the Presburger arithmetic, allowing the extraction of

$$\lambda[x : \mathbf{nat}].\, f\, x \doteq \lambda[x : \mathbf{nat}].\, f\, (x \dot{+} 2)$$

would later require to decide any statement of the form

$$\mathcal{T} \vDash (\forall x.\, f(x) = f(x+2)) \Rightarrow t = u,$$

which is well known to be impossible. Our assumptions on $\mathcal{O}^+$ and $\mathcal{O}^-$ will come later, when proving meta-theoretical properties of CCIC and describing the type-checking algorithm.

We also assume a set of CCIC terms $\mathcal{P}^+$. This set is used later to restrict the types of variables of $\mathcal{X}^+_\star$.

We now define the set of CCIC *well-sorted* terms. Well-sorted are s.t. β-reduction always replaces variables of $\mathcal{X}^-_\star$ and $\mathcal{X}^-_\square$ by terms of $\mathcal{O}^+$ and $\mathcal{P}^+$. Hence, we must forbid pseudo-terms which contain subterms of the form $(\lambda[x :^a U].\, t)\, u$ with $x \in \mathcal{X}^-_\star$ and $u \notin \mathcal{O}^+$ (resp. $x \in \mathcal{X}^-_\square$ and $u \notin \mathcal{P}^+$). Since, as in $\mathrm{CC}_\mathbb{N}$, we do not want terms of the form $(\lambda[x :^r U].\, t)\, u$ and since, we want well-sorted terms to be stable by reduction, we obtain the following definition:

Definition 3.33 — **Well-sorted terms**

A CCIC pseudo-terms t is well-sorted if:

- *t does not contain, on the right of an application or in the branch of a recursor, an unapplied subterm of the form $\lambda[x :^a U].\, t$ with $x \in \mathcal{X}^-_\star \cup x \in \mathcal{X}^-_\square$ or $a = r$.*
- *If t contains a subterm of the form $(\lambda[x :^a U].\, t)\, u$, then $a = u$ and if $x \in \mathcal{X}^-_\star$ (resp. $x \in \mathcal{X}^-_\square$) then $u \in \mathcal{O}^+$ (resp. $u \in \mathcal{P}^+$).*

Moreover, if $C(X) = \forall(\overrightarrow{x_i : U_i}).\, X\, \vec{u}$ is a constructor type, the x_i's and X are not element of $\mathcal{X}^-_\star \cup \mathcal{X}^-_\square$

From now on, we only consider well-sorted terms.

See Chapter 6 for a discussion about theses restrictions.

Note 5

The definitions of *free variable*, *capture-free substitution* and $\beta\iota$-*reduction* do not change from CIC to CCIC.

As usual, it is possible to define layered syntactic classes for CCIC:

Definition 3.34 **Syntactic classes**

The pairwise disjoint syntactic classes of CCIC called objects *($\mathcal{O}$),* predicates *($\mathcal{P}$),* kinds *($\mathcal{K}$), and $\square$ are defined in Figure 3.4.*

This enumeration defined a post-fixed successors function $+1$ on classes ($\mathcal{O}+1=\mathcal{P}$, $\mathcal{P}+1=\mathcal{K}$, $\mathcal{K}+1=\square$, $\square+1=\bot$). We also define $\mathrm{Class}(t)=\mathcal{D}$ *if $t\in\mathcal{D}$ and $\mathcal{D}\in\{\mathcal{O},\mathcal{P},\mathcal{K},\square\}$, and* $\mathrm{Class}(t)=\bot$ *otherwise.*

$$
\begin{aligned}
\mathcal{O} &::= \mathcal{X}^{\star} \mid f\in\underline{\Sigma} \mid \mathcal{O}\,\mathcal{O} \mid \mathcal{O}\,\mathcal{P} \mid \mathrm{Eq}_{\mathcal{P}}(\mathcal{O}) \mid \mathrm{Leib} \mid \mathcal{P}^{[i]} \mid \lambda[x^{\star}:^{a}\mathcal{P}].\,\mathcal{O} \mid \lambda[x^{\square}:^{a}\mathcal{K}].\,\mathcal{O}\\
&::= \mathrm{Elim}(\mathcal{O}:\mathcal{P}\,[\vec{\mathcal{O}}]\to\mathcal{P})\{\vec{\mathcal{O}}\}\\
\mathcal{P} &::= \mathcal{X}^{\square} \mid \sigma\in\underline{\Lambda} \mid \doteq \mid \mathrm{Ind}(X:\mathcal{K})\{\vec{\mathcal{P}}\} \mid \mathcal{P}\,\mathcal{O} \mid \mathcal{P}\,\mathcal{P} \mid \lambda[x^{\star}:^{a}\mathcal{P}].\,\mathcal{P} \mid \lambda[x^{\square}:^{a}\mathcal{K}].\,\mathcal{P}\\
&::= \mathrm{Elim}(\mathcal{O}:\mathcal{P}\,[\vec{\mathcal{O}}]\to\lambda[\overrightarrow{\cdots}].\,\mathcal{K})\{\vec{\mathcal{P}}\} \mid \forall(x^{\star}:^{a}\mathcal{P}).\,\mathcal{P} \mid \forall(x^{\square}:^{a}\mathcal{K}).\,\mathcal{P}\\
\mathcal{K} &::= \star \mid \forall(x^{\star}:^{a}\mathcal{P}).\,\mathcal{K} \mid \forall(x^{\square}:^{a}\mathcal{K}).\,\mathcal{K}\\
\square &::= \square
\end{aligned}
$$

Figure 3.4: CCIC terms classes

Typing judgment

Definition 3.35 **Pseudo-contexts of CCIC**

The typing environments of CIC are defined as $\Gamma,\Delta ::= [] \mid \Gamma,[x:^{a}T]$ s.t. a variable cannot appear twice in the left-hand side of a colon. Moreover, if $x\in\mathcal{X}^{-}_{\star}$, then we require T to be in $\mathcal{P}^{+}$. We use $\mathrm{dom}(\Gamma)$ *for the domain of Γ and $x\Gamma$ for the type associated to x in Γ.*

We can now define the CCIC typing judgment $\Gamma\vdash t:T$. The rules defining $\Gamma\vdash t:T$ are a mix of the ones of $\mathrm{CC}_{\mathbb{N}}$ (notably for the [App] rule side conditions) and the CIC ones (for inductive types):

Definition 3.36 **Typing judgement**

The typing judgment $\Gamma\vdash t:T$ is defined by the rules of Figures 3.5 and 3.6, where

- $\to$ *is a rewriting relation on CCIC, including $\xrightarrow{\beta\iota}$-reduction. Its precise definition is given later.*
- *For any typing environment Γ, $\sim_{\Gamma}$ is the conversion relation of CCIC under environment Γ. As for $\to$, its precise definition is given later.*
- $\mathcal{WT}$ *is a set of terms. Again, its precise definition is given later.*

$$\frac{}{\vdash \star : \square}\ [\text{Ax-1}] \qquad \frac{f \in \underline{\Sigma} \cup \underline{\Lambda} \quad \vdash \tau_f : s}{\vdash f : \tau_f}\ [\text{Symb}]$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash \mathrm{Eq}_T(t) : t \doteq_T t}\ [\text{Eq-Intro}]$$

$$\frac{\begin{array}{c}\vdash \tau_{\mathrm{Leib}} : s \\ \tau_{\mathrm{Leib}} = \forall(T : \star)(t_1\ t_2 : T).\ t_1 \doteq t_2 \to \forall(p : T \to \star).\ p\ t_1 \to p\ t_2\end{array}}{\vdash \mathrm{Leib} : \tau_{\mathrm{Leib}}}\ [\text{Leib}]$$

$$\frac{\Gamma \vdash T : s_T \quad \Gamma, [x :^a T] \vdash U : s_U}{\Gamma \vdash \forall(x :^a T).\ U : s_U}\ [\text{Prod}]$$

$$\frac{\begin{array}{c}\Gamma \vdash t : \forall(x :^u U).\ V \quad \Gamma \vdash u : U \quad u \in \mathcal{WT} \\ \text{if } x \in \mathcal{X}^-_\star, \text{ then } u \text{ must be in } \mathcal{O}^+ \\ \text{if } x \in \mathcal{X}^-_\square, \text{ then } u \text{ must be in } \mathcal{P}^+\end{array}}{\Gamma \vdash t\,u : V\{x \mapsto u\}}\ [\text{App}]$$

$$\frac{\Gamma \vdash \forall(x :^a T).\ U : s \quad \Gamma, [x :^a T] \vdash u : U}{\Gamma \vdash \lambda[x :^a T].\ u : \forall(x :^a T).\ U}\ [\text{Lam}]$$

$$\frac{\Gamma \vdash V : s \quad \Gamma \vdash t : T \quad s \in \{\star, \square\} \quad x \in \mathcal{X}^s \setminus \mathrm{dom}(\Gamma)}{\Gamma, [x : V] \vdash t : T}\ [\text{Weak}]$$

$$\frac{x \in \mathrm{dom}(\Gamma) \cap \mathcal{X}^{s_x} \quad \Gamma \vdash x\Gamma : s_x}{\Gamma \vdash x : x\Gamma}\ [\text{Var}]$$

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s' \quad T \sim_\Gamma T'}{\Gamma \vdash t : T'}\ [\text{Conv}]$$

Figure 3.5: CCIC Typing Rules (CC rules)

Notation. A term t is *well-formed* (or *well-typed*), denoted by $\Gamma \vdash t$, if there exists T s.t. $\Gamma \vdash t : T$. A typing environment is *well-formed*, denoted by $\Gamma \vdash$, if there exists a term t s.t. $\Gamma \vdash t$.

Definition 3.37

Among well-typed terms, we distinguish:

- *The set* $\mathbb{O}(\Gamma)$ *of* well-typed objects *under* Γ*, i.e. the set composed of terms* t *s.t.* $\Gamma \vdash t : T$ *and* $\Gamma \vdash T : \star$ *for some* T*.*
- *The set* $\mathbb{P}(\Gamma)$ *of* well-typed predicates *under* Γ*, i.e. the set of terms* T *s.t.* $\Gamma \vdash T : K$ *and* $\Gamma \vdash K : \square$ *for some* K*.*
- *The set* $\mathbb{K}(\Gamma)$ *of* well-typed predicate types *under* Γ*, i.e. the set of terms* T *s.t.* $\Gamma \vdash T : \square$*.*

We also define the following sets:

$$\frac{A = \forall(\overrightarrow{x : T}).\star \quad \vdash A : \Box \quad \text{for all } i,\ \Gamma \vdash C_i(X) : \star \quad \text{for all } i,\ C_i(X) \text{ is a strictly positive constructor in } X \quad I = \mathrm{Ind}(X : A)\{\overrightarrow{C_i(X)}\} \text{ is in } \to\text{-normal form}}{\Gamma \vdash I : A} \ [\textsc{Ind}]$$

$$\frac{I = \mathrm{Ind}(X : T)\{\overrightarrow{C_i(X)}\} \quad \Gamma \vdash I : T}{\Gamma \vdash I^{[k]} : C_k(X)\{X \mapsto I\}} \ [\textsc{Constr}]$$

$$\frac{A = \forall(\overrightarrow{x : U}).\star \quad I = \mathrm{Ind}(X : A)\{\overrightarrow{C_j(X)}\} \quad \Gamma \vdash I : A \quad \vdash Q : \forall(\overrightarrow{x : U}).\,(I\ \vec{x}) \to \star \quad \text{for all } i,\ T_i = \Delta^{\star}\{I, X, C_i(X), Q, I^{[i]}\} \quad \vdash T_i : \star \quad \Gamma \vdash f_i : T_i \quad \text{for all } j,\ \Gamma \vdash a_j : A_j\{\overrightarrow{x \mapsto a}\} \quad \Gamma \vdash c : I\ \vec{a}}{\Gamma \vdash \mathrm{Elim}(c : I\,[\vec{a}] \to Q)\{\vec{f}\} : Q\ \vec{a}\ c} \ [\textsc{Elim-}\star]$$

$$\frac{A = \forall(\overrightarrow{x : U}).\star \quad I = \mathrm{Ind}(X : A)\{\overrightarrow{C_j(X)}\} \text{ is small} \quad Q = \forall(\overrightarrow{x : U})(y : I\ \vec{x}).\,K \text{ is in } \to\text{-normal form} \quad [\overrightarrow{x : U}], [y : I\ \vec{x}] \vdash K : \Box \quad \text{for all } i,\ T_i = \Delta^{\Box}\{I, X, C_i(X), \vec{x}\, y, K, I^{[i]}\} \quad \vdash T_i : \Box \quad \Gamma \vdash f_i : T_i \quad \text{for all } j,\ \Gamma \vdash a_j : A_j\{\overrightarrow{x \mapsto a}\} \quad \Gamma \vdash c : I\ \vec{a}}{\Gamma \vdash \mathrm{Elim}(c : I\,[\vec{a}] \to Q)\{\vec{f}\} : K\{\overrightarrow{x \mapsto}, y \mapsto c\}} \ [\textsc{Elim-}\Box]$$

Figure 3.6: CCIC Typing Rules (Inductive Types)

$$\mathbb{O} = \bigcup\{\mathbb{O}(\Gamma) \mid \Gamma \textit{ s.t. } \Gamma \vdash\}$$
$$\mathbb{P} = \bigcup\{\mathbb{P}(\Gamma) \mid \Gamma \textit{ s.t. } \Gamma \vdash\}$$
$$\mathbb{K} = \bigcup\{\mathbb{K}(\Gamma) \mid \Gamma \textit{ s.t. } \Gamma \vdash\}$$

We now define a notion of well-formed substitution. In the Calculus of Constructions, a substitution θ is well-formed from a typing environment Γ to a typing environment Δ if for all $x \in \mathrm{dom}(\theta)$, $\Delta \vdash x\theta : x\Gamma$. One can then easily prove $\Delta \vdash t\theta : T\theta$ if $\Gamma \vdash t : T$ - a property called *stability by substitution*.

Assume now that Γ is a typing environment of the form $\Gamma_1, [x :^r a \doteq b], \Gamma_2$ (a and b being two terms in the set of extractable terms $\mathcal{O}^+$). Stability by substitution claims that if we have a derivation of the form $\Gamma \vdash t : T$, then we can substitute x by a any term P (s.t. $\Gamma \vdash P : a \doteq b$) in the derivation $\Gamma \vdash t : T$ and obtain a proof of $\Gamma_1, \Gamma_2\theta \vdash t\theta : T\theta$, where $\theta = \{x \mapsto P\}$. However, this is not true in general since the equation $a \doteq b$ is then removed from context and not usable in the definition of $\sim_{\Gamma_1, \Gamma_2\theta}$. Hence, $a\theta \sim_{\Gamma_1, \Gamma_2\theta} b\theta$ does not hold in general, and $\Gamma_1, \Gamma_2\theta \vdash t\theta : T\theta$ is not necessarily derivable.

Another problems arise when substituting variables in extractable equations. Suppose now that $\Gamma \vdash \lambda[p :^r a \doteq b].u$ with a and b in $\mathcal{O}^+$. Stability by substitution now leads to $\Delta \vdash \lambda[p :^r a\theta \doteq b\theta].u\theta$. Again, for this result to hold, we need $a\theta$ and $b\theta$ to be extractable, i.e. $\mathcal{O}^+$ has to be stable by application of a well-formed substitution.

Hence the following definition of well-formed substitutions:

Definition 3.38 **Well-sorted and well-formed substitution**

A substitution θ *is* well-sorted *from* Γ *to* Δ*, written* $\theta : \Gamma \hookrightarrow \Delta$*, if*

- $\forall x \in \mathrm{dom}(\theta)$, x *is not annotated by* r *in* Γ,
- *if* x *is annotated by* r *is* Γ*, then* $x \in \mathrm{dom}(\Delta)$, x *is annotated by* r *in* Δ*, and* $x\Delta = x\Gamma\theta$.
- $\forall x \in \mathrm{dom}(\theta)$, *if* $x \in \mathcal{X}^-_\star$ *then* $x\theta \in \mathcal{O}^+$,
- $\forall x \in \mathrm{dom}(\theta)$, *if* $x \in \mathcal{X}^-_\square$ *then* $x\theta \in \mathcal{P}^+$,
- $\forall x \in \mathrm{dom}(\theta)$. $\mathrm{Class}(x) = \mathrm{Class}(x\theta)$.
- $\forall x \in \mathrm{dom}(\theta)$. $x\theta$ *is a weak term.*

Moreover, if for all $x \in \mathrm{dom}(\theta)$, $\Delta \vdash x\theta : x\Gamma\theta$*, then we say that* θ *is* well-formed*, written* $\theta : \Gamma \rightsquigarrow \Delta$.

Incorporation of a PMS algebra into CCIC

We start out incorporating of a parametric theory into CCIC. From now on, let $\Lambda_\mathcal{E}$ be a sort signature and Σ a $\Lambda_\mathcal{E}$-signature.

Translation of sort constructors and function symbols

We introduce into CCIC two new sets of symbols: a set of type level symbols $\underline{\Lambda} = \{\underline{\sigma} \mid \underline{\sigma} \in \Lambda\}$, and a set of object level symbols $\underline{\Sigma} = \{\underline{f} \mid f \in \Sigma\}$. We then assign for any object (resp. type) level symbol a type level (resp. kind level) CCIC terms denoting the translation of their arities.

As an example, for the signature of Presburger arithmetic $\Lambda = \{\mathbf{nat}\}$ and $\Sigma = \{\mathbf{0} : \mathbf{nat}, \mathbf{S} : \mathbf{nat} \to \mathbf{nat}, + : \mathbf{nat} \times \mathbf{nat} \to \mathbf{nat}\}$, we then introduce 4 new symbols with the following types:

$$\begin{array}{ll} \underline{\mathbf{nat}} & : \star \\ \underline{\mathbf{0}} & : \underline{\mathbf{nat}} \\ \underline{\mathbf{S}} & : \underline{\mathbf{nat}} \to \underline{\mathbf{nat}} \\ \underline{+} & : \underline{\mathbf{nat}} \to \underline{\mathbf{nat}} \to \underline{\mathbf{nat}} \end{array}$$

Non surprisingly, **nat** being a non-parametric first-order sort constructor, we associate the CCIC-sort $\star$ to the symbol $\underline{\mathbf{nat}}$. The types associated to the function symbols are simply the curryfied versions of their arities (replacing **nat** by $\underline{\mathbf{nat}}$).

In the case of a parametric signature, like the one for parametric lists, $\Lambda = \{\mathbf{list}/1\}$ and $\Sigma = \{\mathbf{nil}, \mathbf{cons}, \mathbf{app}\}$ with

$$\begin{array}{lll} \mathbf{nil} & : \forall\alpha. & \to \mathbf{list}(\alpha) \\ \mathbf{cons} & : \forall\alpha.\, \alpha \times \mathbf{list}(\alpha) & \to \mathbf{list}(\alpha) \\ \mathbf{app} & : \forall\alpha.\, \mathbf{list}(\alpha) \times \mathbf{list}(\alpha) & \to \mathbf{list}(\alpha) \end{array}$$

the translation is similar, but the sort variables appearing in the arities are now abstracted with CCIC-variables of sort $\square$, i.e.:

$$
\begin{array}{ll}
\underline{\textbf{list}} & : \star \to \star \\
\underline{\textbf{nil}} & : \forall(T : \star).\, \underline{\textbf{list}}\, T \\
\underline{\textbf{cons}} & : \forall(T : \star).\, T \to (\underline{\textbf{list}}\, T) \to (\underline{\textbf{list}}\, T) \\
\underline{\textbf{app}} & : \forall(T : \star).\, (\underline{\textbf{list}}\, T) \to (\underline{\textbf{list}}\, T) \to (\underline{\textbf{list}}\, T)
\end{array}
$$

Definition 3.39 **Signature translation**

Let $\Lambda_{\mathcal{E}}$ be a set of sorts and Σ a $\Lambda_{\mathcal{E}}$-signature. A CCIC-translation of Σ is given by:

- *a mapping $\hat{\cdot}$ from sort variables to CCIC type level variables (i.e. we associate to any sort variable $\alpha \in \mathcal{E}$ a unique CCIC type level variable $\hat{\alpha} \in \mathcal{X}^{\square}$).*
- *a set of CCIC sort constructor symbols (with associated CCIC types) $\underline{\Lambda} = \{\underline{\sigma} : \tau_{\underline{\sigma}} \mid \sigma \in \Lambda\}$ where $\tau_{\underline{\sigma}} = \underbrace{\star \to \cdots \star \to}_{\text{arity}(\underline{\sigma}) \text{ times}} \star$,*
- *a set of CCIC function symbols (with associated CCIC types) $\underline{\Sigma} = \{\underline{\sigma} : \tau_{\underline{f}} \mid f \in \Sigma\}$ where $\tau_{\underline{f}}$ is defined as follow:*

$$\tau_{\underline{f}} = \forall(\overrightarrow{\hat{\alpha} : \star})_{\alpha \in \text{FV}(\text{arity}(f))}.\, [\![\tau_1]\!] \to \cdots [\![\tau_n]\!] \to [\![\sigma]\!]$$

with $\text{arity}(f) = \tau_1 \times \cdots \tau_n \to \sigma$ *and* $[\![\cdot]\!] : \Lambda_{\mathcal{E}} \to$ *CCIC inductively defined as*

$$
\begin{aligned}
[\![\alpha]\!] &= \hat{\alpha} \\
[\![\underline{\sigma}(\tau_1, \ldots, \tau_n)]\!] &= \underline{\sigma}\, [\![\tau_1]\!] \ldots [\![\tau_n]\!]
\end{aligned}
$$

We have the immediate result:

Fact 3.40

For any CCIC sort constructor symbol $\underline{\sigma}$ (resp. CCIC function symbol $\underline{f}$), we have $\vdash \tau_{\underline{\sigma}} : \square$ (resp. $\vdash \tau_{\underline{f}} : \star$).

Translation of Σ-rewriting systems

We now move to the translation of a Σ-rewriting system.

In parametric multi-sorted algebra, a rewrite system preserves sorts as both sides of any of its rule have the same sort. The analog for type systems is that a rewrite rule $l \to r$ preserves types if there exists an environment Γ and a term T s.t. $\Gamma \vdash l : T$ and $\Gamma \vdash r : T$ - we say that $l \to r$ is well-formed under Γ.

When translating a first-order rule into CCIC, one has to give the translation of the rewrite rule into a CCIC rewrite rule $l \to r$ and a CCIC typing environment Γ s.t. $l \to r$ is well formed under Γ.

Taking Presburger arithmetic as example, the following rewriting system

$$
\begin{aligned}
\forall x : \textbf{nat}. \quad & x + 0 \to x \\
\forall x, y : \textbf{nat}. \quad & x + S(y) \to S(x + y)
\end{aligned}
$$

is translated into

$$\begin{aligned} x \dot{+} \mathbf{0} &\to x & \Gamma &= [x : \underline{\mathbf{nat}}] \\ x \dot{+} (\mathbf{S}\, y) &\to \mathbf{S}\, (x \dot{+} y) & \Gamma &= [x : \underline{\mathbf{nat}}], [y : \underline{\mathbf{nat}}] \end{aligned}$$

It is clear that all these rewrite rules are well formed under the typing environments written on their right.

The case of parametric rules is not harder. For example, the definition of concatenation of parametric lists

$$\begin{aligned} \forall \alpha.\, \forall l : \mathbf{list}(\alpha). & & \mathbf{app}(l, \mathbf{nil}) &\to l \\ \forall \alpha.\, \forall l\, l' : \mathbf{list}(\alpha), x : \alpha. & & \mathbf{app}(l, \mathbf{cons}(x, l')) &\to \mathbf{cons}(x, \mathbf{app}(l, l')) \end{aligned}$$

is translated into

$$\begin{aligned} \underline{\mathbf{app}}\, A\, l\, (\underline{\mathbf{nil}}\, A) &\to l & \Gamma &= [A : \star], [l : \underline{\mathbf{list}}\, A] \\ \underline{\mathbf{app}}\, A\, l\, (\underline{\mathbf{cons}}\, A\, x\, l') &\to \underline{\mathbf{cons}}\, A\, (\underline{\mathbf{app}}\, A\, l\, l') & \Gamma &= [A : \star], [l\, l' : \underline{\mathbf{list}}\, A], [x : A] \end{aligned}$$

Definition 3.41 — **Rewrite rule translation**

Let $q = \forall \vec{\alpha}.\, \forall x_1 : \tau_1 \cdots x_n : \tau_n.\, l \to r$ *be a* Σ*-rewrite rule. We write* $\mathcal{I}_S$ *for the sort assignment* $[x_1 : \tau_1], \ldots, [x_n : \tau_n]$*. The translation of* q *is* $\Gamma : (\!|l|\!) \to (\!|r|\!)$ *where:*

- $(\!|\cdot|\!)$ *is inductively defined as*

$$\begin{aligned} (\!|x|\!) &= x \\ (\!|f(t_1, \ldots, t_n)|\!) &= \underline{f}\, [\![\beta_1 \xi]\!] \cdots [\![\beta_k \xi]\!]\, (\!|t_1|\!) \cdots (\!|t_n|\!) \end{aligned}$$

  where i) f *has arity* $\forall \beta_1, \ldots \beta_k.\, \mu_1 \times \cdots \times \mu_n \to \mu$ *ii)* $\mathcal{I}_S.\, f(t_1, \ldots, t_n)$ *is a term scheme of sort* σ *and the* $\mathcal{I}_S.\, t_i$ *are term schemes of respective sorts* σ_i*, and iii)* ξ *is the sort substitution of domain* $\{\beta_1, \ldots, \beta_k\}$ *s.t.* $\forall i.\, \sigma_i = \mu_i \xi$ *and* $\sigma = \mu \xi$*.*

- $\Gamma = [A_1 : \star], \ldots, [A_p : \star], [x_1 : [\![\tau_1]\!]], \ldots, [x_n : [\![\tau_n]\!]]$
  with $\{A_1, \ldots, A_n\} = (\mathrm{FV}((\!|l|\!)) \cup \mathrm{FV}((\!|r|\!))) \cap \mathcal{X}^{\square}$*.*

Lemma 3.42

For any Σ-rewrite rule $l \to r$, $\Gamma : (\!|l|\!) \to (\!|r|\!)$ is a well formed rewrite rule under Γ.

Proof. If $l \to r = \forall \overline{\alpha}.\, \forall x_1 : \tau_1 \cdots x_n : \tau_n.\, l \to r$ be a Σ-rewrite rule, we prove by induction on $(\!|l|\!)$ and $(\!|r|\!)$, that $\Gamma \vdash (\!|l|\!) : [\![\sigma]\!]$ and $\Gamma \vdash (\!|r|\!) : [\![\sigma]\!]$ where σ is the common sort of l and r. □

One major drawback of using such a translation is that, in the case of parametric rewrite rules, we obtain non-left linear rewrite rules, which complicates the confluence proof.

This can be overcomed by using the notion of well typed rule of [4], where a substitution applies to the left hand side before typing it.

Instead of defining **app** as in our previous example, one can give the following rewriting system instead:

$$\begin{aligned} \underline{\mathbf{app}}\, A\, l\, (\underline{\mathbf{nil}}\, A) &\to l & \Gamma &= [A : \star], [l : \underline{\mathbf{list}}\, A] \\ \underline{\mathbf{app}}\, A_1\, l\, (\underline{\mathbf{cons}}\, A_2\, x\, l') &\to \underline{\mathbf{cons}}\, A\, (\underline{\mathbf{app}}\, A\, l\, l') & \Gamma &= [A : \star], [l\, l' : \underline{\mathbf{list}}\, A], [x : A] \end{aligned}$$

along with the substitution $\rho = \{A_1 \mapsto A, A_2 \mapsto A\}$ for the second rewriting rule.

Definition 3.43 — **Well typed rule [4]**

A rewrite rule $l \to r$ *with* $l = f\ \vec{t}$ *and* $f : \forall(\overrightarrow{x : T}).\, U$ *is well typed is there exists an environment* Γ *and a substitution* ρ *s.t.*

- $\Gamma \vdash l\rho : U\theta\rho$ *and* $\Gamma \vdash r : U\theta\rho$,
- $\mathrm{dom}(\rho) \cap \mathrm{dom}(\Gamma) = \varnothing$,
- *for any* Δ, δ, T, *if* $\Delta \vdash l\delta : T$ *then* $\delta : \Gamma \rightsquigarrow \Delta$,
- *for any* Δ, δ, T, *if* $\Delta \vdash l\delta : T$ *then* $\delta \downarrow \rho\delta$.

where $\theta = \{\overrightarrow{x \mapsto t}\}$.

We write $(\Gamma, \rho) : l \to r$ *when these conditions are satisfied.*

Linearisation of translated rules can be done as follow:

Definition 3.44

Let $\Gamma : l \to r$ *be the translation of a* Σ*-rewrite rule. For any* $p \in \mathrm{Pos}(l)$ *s.t.* $l_{|p} \in \mathcal{X}^{\square}$, *let* A_p *be fresh variables of* $\mathcal{X}^{\square}$. *We define* θ *and* ρ *as follow:*

$$\theta = [p \leftarrow A_p \mid p \in \mathrm{Pos}(l), l_{|p} \in \mathcal{X}^{\square}]$$
$$\rho = \{A_p \to l_{|p} \mid p \in \mathrm{Pos}(l), l_{|p} \in \mathcal{X}^{\square}\}$$

The linearisation *of* $\Gamma : l \to r$ *is the rewrite rule* $l\theta \to r$, *along with the environment* Γ *and the substitution* ρ.

Conversion relation

From now on, let $\to$ be a rewriting system, defined as the union of $\xrightarrow{\beta\iota}$ and $\xrightarrow{R}$ where R is the translation of a Σ-rewriting system.

We are now left to define our conversion relation $\sim_\Gamma$. The two main differences with $\mathrm{CC}_{\mathbb{N}}$ are the following.

- Our notion of algebraisation now works modulo the expected sort of the resulting first-order term. This is the aim of the next section.
- A notion of *weak* terms is introduced in Section 3.3 so that conversion operates only on them - the others being only converted with $\to$-reductions. This is needed to forbid inconsistencies at object level to be lift up to the type level.

We start with our new notion of algebraisation.

Algebraisation

Algebraisation is the first part of the hypotheses extraction: it allows transforming a CCIC term into its first-order counterpart. We illustrate the difficulties with examples.

The case for pure algebraic terms is as for $CC_{\mathbb{N}}$. The definition becomes a little harder for parametric signatures. The theory of lists gives us a simple example. From the definition of conversion of an algebra to CCIC, we know than the **app** symbol has type

$$\forall(A : \star).\, \underline{\mathbf{list}}\, A \to \underline{\mathbf{list}}\, A \to \underline{\mathbf{list}}\, A$$

Thus, a fully applied, well formed term having the symbol **app** at head position is necessarily of the form $\mathbf{app}\, T\, l\, l'$, T being the type of the elements of the list. Algebraisation of such a term will erase all the type parameters: in our example, $\mathcal{A}(\mathbf{app}\, T\, l\, l') = \mathbf{app}(\mathcal{A}(l), \mathcal{A}(l'))$.

Our conversion being defined on non well-formed terms, we now come to the case of algebraisation of non-pure algebraic terms or even ill-formed terms. As for $CC_{\mathbb{N}}$, for non-pure algebraic terms, the problem is solved by abstracting non-algebraic subterms with fresh variables. For example, algebraisation of $1 + t$ with t non-algebraic yields $1 + x_{\mathbf{nat}}$ where $x_{\mathbf{nat}}$ is an abstraction variable of sort **nat** for t. The problem is harder for:

- *parametric symbols*: in $(\mathbf{cons}\, T\, t\, (\mathbf{nil}\, U))$ with t non algebraic, should t be abstracted by a variable of sort **nat** or $\mathbf{list}(\mathbf{nat})$?
- *ill-formed terms*: should $(\mathbf{cons}\, T\, 0\, (\mathbf{cons}\, T\, (\mathbf{nil}\, U)\, (\mathbf{nil}\, T)))$ be abstracted as a list of natural numbers or as a list of lists ?

The solution adopted here is to postpone the decision: $\mathcal{A}(t)$ is defined as a function from $\Lambda_{\mathcal{E}}$ to the terms of $\mathcal{T}$ s.t. $\mathcal{A}(t)(\sigma)$ is the algebraisation of t under the condition that t is a CCIC representation of a first-order term of sort σ.

We now give the formal definition of $\mathcal{A}(\cdot)$.

Let $\{\mathcal{Y}_\sigma\}_\sigma$ be a $\Lambda_{\mathcal{E}}$-sorted family of pairwise disjoint countable infinite sets of variables of sort σ. Let $\mathcal{Y} = \bigcup_\sigma \mathcal{Y}_\sigma$.

For any $x \in \mathcal{X}^\star$ and sort $\sigma \in \Lambda_{\mathcal{E}}$, let x^σ be a fresh first-order variable of sort σ. We denote by $\mathcal{Z}_\sigma$ the set $\{x^\sigma \mid x \in \mathcal{X}\}$ and by $\mathcal{Z}$ the set $\bigcup_\sigma \mathcal{Z}_\sigma$.

For any equivalence relation $\mathcal{R}$ and sort $\sigma \in \Lambda_{\mathcal{E}}$, we suppose the existence of a function $\pi^\sigma_{\mathcal{R}} : \mathrm{CCIC} \to \mathcal{Y}_\sigma$ s.t. $\pi^\sigma_{\mathcal{R}}(t) = \pi^\sigma_{\mathcal{R}}(u)$ if and only if $t\, \mathcal{R}\, u$ (i.e. $\pi^\sigma_{\mathcal{R}}(t)$ is the element of $\mathcal{Y}_\sigma$ representing the class of t modulo $\mathcal{R}$).

Definition 3.45 — **Well applied term**

A term is well-applied *if it is of the form* $f\, [\overrightarrow{T_\alpha}]_{\alpha \in \vec{\alpha}}\, t_1 \cdots t_n$ *with* $f : \forall \vec{\alpha}.\, \sigma_1 \times \cdots \times \sigma_n \to \sigma$.

Example 3.46

Example of well applied terms are $\mathbf{0}$, $\mathbf{S}\, t$, or $\mathbf{app}\, T\, l\, l'$ - T being the type parameter. Note that we do not require the terms to be well formed.

Note 6

When writing that $f\, \vec{T}\, \vec{t}$ is well applied, we implicitly require that $\vec{T}$ contains all the type parameters and only them.

Definition 3.47 — **Algebraisation**

Assume $t \in CCIC$ *and an equivalence relation* $\mathcal{R}$. *The* algebraisation of t modulo $\mathcal{R}$ *is the function* $\mathcal{A}_{\mathcal{R}}(t) : \Lambda_{\mathcal{E}} \to \mathcal{T}(\mathcal{Y} \cup \mathcal{Z})$ *defined as:*

$$\begin{aligned}
\mathcal{A}_{\mathcal{R}}(x)(\sigma) &= x^{\sigma} && \text{if } x \in \mathcal{X}^{\star} \\
\mathcal{A}_{\mathcal{R}}(f\,\vec{T}\,[u_i]_{i\in n})(\tau\xi) &= f(\mathcal{A}_{\mathcal{R}}(u_1)(\sigma_1\xi), \ldots, \mathcal{A}_{\mathcal{R}}(u_n)(\sigma_n\xi)) && \text{if } (*) \\
\mathcal{A}_{\mathcal{R}}(t)(\tau) &= \Pi^{\tau}_{\mathcal{R}}(t) && \text{otherwise}
\end{aligned}$$

where $(*)$ |
i) $f\,\vec{T}\,[u_i]_{i\in n}$ *is well applied,*
ii) f *is of arity* $\forall\vec{\alpha}.\,\sigma_1 \times \cdots \sigma_n \to \tau$,
iii) ξ *is a* $\Lambda_{\mathcal{E}}$*-substitution.*

For any relation R, $\mathcal{A}_R$ *is defined as* $\mathcal{A}_{\mathcal{R}}$ *where* $\mathcal{R}$ *is the smallest equivalence relation containing* R. *We call* σ-alien *(or* alien *for short) the subterms of* t *abstracted by a variable in* $\mathcal{Y}_\sigma$. *A term is* algebraically pure *(or* pure *for short) if it does not contain aliens.*

Example 3.48

Let $t \equiv \mathbf{cons}\,T\,\mathbf{0}\,(\mathbf{cons}\,U\,(\mathbf{nil}\,V)\,(\mathbf{nil}\,U))$ and R a relation on the terms of CCIC. Then, assuming $x_{\mathbf{nat}}, y_{\mathbf{list}}, z_{\mathbf{nat}}$ being abstraction variables, we have

$$\begin{aligned}
\mathcal{A}_R(t)(\mathbf{list}(\mathbf{nat})) &= \mathbf{cons}(\mathcal{A}_R(\mathbf{0})(\mathbf{nat}), \mathcal{A}_R(\mathbf{cons}\,U(\mathbf{nil}\,V)\,(\mathbf{nil}\,U))(\mathbf{list}(\mathbf{nat}))) \\
&= \mathbf{cons}(0, \mathbf{cons}(\mathcal{A}_R(\mathbf{nil}\,V)(\mathbf{nat}), \mathcal{A}_R(\mathbf{nil}\,U)(\mathbf{list}(\mathbf{nat})))) \\
&= \mathbf{cons}(0, \mathbf{cons}(x_{\mathbf{nat}}, \mathbf{nil}))
\end{aligned}$$

whereas

$$\begin{aligned}
\mathcal{A}_R(t)(\mathbf{list}(\sigma)) &= \mathbf{cons}(\mathcal{A}_R(\mathbf{0})(\sigma), \mathcal{A}_R(\mathbf{cons}\,U(\mathbf{nil}\,V)\,(\mathbf{nil}\,U))(\mathbf{list}(\sigma))) \\
&= \mathbf{cons}(y_{\mathbf{list}}, \mathbf{cons}(\mathcal{A}_R(\mathbf{nil}\,V)(\sigma), \mathcal{A}_R(\mathbf{nil}\,U)(\mathbf{list}(\sigma)))) \\
&= \mathbf{cons}(y_{\mathbf{list}}, \mathbf{cons}(\mathbf{nil}, \mathbf{nil}))
\end{aligned}$$

and, $\mathcal{A}_R(t)(\mathbf{nat}) = z_{\mathbf{nat}}$.

Note that, as explained before, the algebraisation does not only depend on the terms being abstracted, but also on the expected sort of the result. This is clearly seen on the example: when abstracting the (heterogeneous and ill-formed) list $\mathbf{0} :: \mathbf{nil} :: \mathbf{nil}$ as a list of lists, $\mathbf{0}$ is seen as an alien and thus abstracted. Conversely, when this list is abstracted as a list of natural numbers, $\mathbf{0}$ is considered algebraic but the $\mathbf{nil}$ element is then seen as an alien and abstracted. Of course, as clear from the last case, if the list is algebraised as a natural number, it gets directly abstracted by a variable.

Lemma 3.49

Let $t \in$ CCIC, $\sigma \in \Lambda_{\mathcal{E}}$ and $\mathcal{R}$ a binary relation on CCIC, then $\mathcal{A}_{\mathcal{R}}(t)(\sigma)$ is a well formed $\mathcal{T}$-term of sort σ.

Proof. By induction on the definition on $\mathcal{A}_R(\cdot)$. □

Weak terms

We distinguish a class of terms called *weak*. This class of terms will play an important role in the following as they restrict the interaction between the conversion at object level and the strong ι-reduction.

An example of what will be a non weak term is

$$t = \lambda[x : \mathbf{nat}].\, \mathrm{Elim}^S(x : \mathbf{nat}\ [] \to Q)\{\mathbf{nat}, \lambda[x : \mathbf{nat}][T : Q\,x].\, \mathbf{nat} \to \mathbf{nat}\}.$$

Such a term is problematic in the sense that when applied to convertible terms, it can βι-reduces to type-level terms that are not βι-convertible. Suppose that the conversion relation given for $CC_{\mathbb{N}}$ is canonically extended to CCIC. We know that there exists a typing environment Γ s.t. $\mathbf{0} \sim_\Gamma \mathbf{S}\,\mathbf{0}$, and hence, by congruence, $t\,\mathbf{0} \sim_\Gamma t\,(\mathbf{S}\,\mathbf{0})$. Now, it is easy to check that $t\,\mathbf{0} \xrightarrow{\beta\iota}_* \mathbf{nat}$ and $t\,(\mathbf{S}\,\mathbf{0}) \xrightarrow{\beta\iota}_* (\mathbf{nat} \to \mathbf{nat})$. It is known that having a relation capturing $\mathbf{nat} \sim_\Gamma \mathbf{nat} \to \mathbf{nat}$ breaks strong normalization of β-reduction, the term $\omega = \lambda[x : \mathbf{nat}].\, x\,x$ being typable in such a system.

On the contrary, *weak* terms are defined s.t. they cannot lift inconsistencies from object level to a higher level.

Definition 3.50 — **Weak terms**

A term is said weak *if it does not contain open elimination (i.e. does not contain a term of the form* $\mathrm{Elim}(t : I\,[\vec{u}] \to Q)\{\vec{f}\}$ *with* t *having free variables).*

A constructor type $C(X) = \forall(\overrightarrow{x_i : U_i}).\, X\,\vec{u}$ *in* X *is* weak *if for any* i, X *does not occur in* U_i *or* $U_i = X\,\vec{v}$. *An inductive type is weak if all its constructors type are weak.*

We denote by $\mathcal{WT}$ *the set of weak terms.*

Conversion relation

We have now all the ingredients to define our conversion relation $\sim_\Gamma$:

Definition 3.51 — **Conversion relation**

Rules of Figure 3.7 defines a family $\{\sim_\Gamma\}$ *of CCIC binary relations, where for each rule of the form*

$$\frac{E_1 \cdots E_n}{t \sim_\Gamma u}$$

one has to read

$$\frac{E_1 \cdots E_n \quad \mathrm{Class}(t) \neq \bot \quad \mathrm{Class}(u) \neq \bot}{t \sim_\Gamma u}$$

Note 7

From $CC_{\mathbb{N}}$, the main differences are:

- The rules [App] and [Elim] are now restricted to weak terms.
- Conversion for terms being destructed by a recursor is restricted to $\leftrightarrow_*$.

$$\frac{}{t \sim_\Gamma t}\ [\textsc{Refl}] \qquad \frac{[x :^r T] \in \Gamma \quad T \to_* t \doteq u \quad t, u \in \mathcal{O}^+}{t \sim_\Gamma u}\ [\textsc{Eq}]$$

$$\frac{\begin{array}{c} E \vDash \mathcal{A}_{\sim_\Gamma}(t)(\tau) = \mathcal{A}_{\sim_\Gamma}(u)(\tau) \quad t, u \in \mathcal{O}^- \\ E = \{\mathcal{A}_{\sim_\Gamma}(w_1)(\sigma) = \mathcal{A}_{\sim_\Gamma}(w_2)(\sigma) \mid w_1 \sim_\Gamma w_2, \sigma \in \Lambda_\mathcal{E}, w_1, w_2 \in \mathcal{O}^+\} \end{array}}{t \sim_\Gamma u}\ [\textsc{Ded}]$$

$$\frac{t \to t' \quad t' \sim_\Gamma u}{t \sim_\Gamma u}\ [\textsc{Rw-Left}] \qquad \frac{u \to u' \quad t \sim_\Gamma u'}{t \sim_\Gamma u}\ [\textsc{Rw-Right}]$$

$$\frac{T \sim_\Gamma U \quad t \sim_{\Gamma,[x:^a T]} u}{\lambda[x :^a T].\, t \sim_\Gamma \lambda[x :^a U].\, u}\ [\textsc{Lam}] \qquad \frac{T \sim_\Gamma U \quad t \sim_{\Gamma,[x:^a T]} u}{\forall(x :^a T).\, t \sim_\Gamma \forall(x :^a U).\, u}\ [\textsc{Prod}]$$

$$\frac{T \sim_\Gamma U \quad t \sim_\Gamma u}{\mathrm{Eq}_T(t) \sim_\Gamma \mathrm{Eq}_U(u)}\ [\textsc{EqT}] \qquad \frac{t_1 \sim_\Gamma u_1 \quad t_2 \sim_\Gamma u_2 \quad t_i, u_i \text{ are weak}}{t_1\, t_2 \sim_\Gamma u_1\, u_2}\ [\textsc{App}^w]$$

$$\frac{\begin{array}{c} I \text{ and } I' \text{ reduce to weak inductive types} \\ \mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \text{ and } \mathrm{Elim}(t' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\} \text{ are weak} \\ t \leftrightarrow_* t' \quad I \sim_\Gamma I' \quad Q \sim_\Gamma Q' \quad \vec{v} \sim_\Gamma \vec{v}' \quad \vec{f} \sim_\Gamma \vec{f}' \end{array}}{\mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \sim_\Gamma \mathrm{Elim}(t' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\}}\ [\textsc{Elim}^w]$$

$$\frac{A \sim_\Gamma A' \quad \overrightarrow{C(X)} \sim_\Gamma \overrightarrow{C'(X)}}{\mathrm{Ind}(X : A)\{\overrightarrow{C(X)}\} \sim_\Gamma \mathrm{Ind}(X : A')\{\overrightarrow{C'(X)}\}}\ [\textsc{Ind}] \qquad \frac{T \sim_\Gamma U \quad n \in \mathbb{N}}{T^{[n]} \sim_\Gamma U^{[n]}}\ [\textsc{Constr}]$$

Figure 3.7: CCIC conversion relation

· An equation is extractable only if it is defined on terms in $\mathcal{O}^+$. Likewise, first-order deduction is now only done on terms of $\mathcal{O}^-$.

CHAPTER 4

Meta-theoretical Properties of CCIC

From now on, let $\mathcal{T}$ be a theory over the parametric multi-sorted signature $(\Lambda_{\mathcal{E}}, \Sigma)$ and R the CCIC translation of a Σ-rewriting system s.t. $\xrightarrow{\beta\iota R}$ is confluent. We write $\to$ for $\xrightarrow{\beta\iota R}$. We also assume that rewrite rules of R preserve syntactic classes.

We start with the road-map, which is basically the one for Pure Type Systems. The main meta-theoretical properties addressed in this chapter are confluence, subject reduction, strong normalization and consistency. As usual, they need proving additional properties (weakening, stability via well-formed substitution, type correction, ...).

Definition 4.1

We now define all the restrictions our sets $\mathcal{O}^+$, $\mathcal{O}^-$ and $\mathcal{P}^+$ must obey:

1. $\mathcal{O}^+ \subseteq \mathcal{O}^- \subseteq \mathcal{O}$,
2. $\mathcal{P}^+ \subseteq \mathcal{P}$,
3. *$\mathcal{O}^+$, $\mathcal{O}^-$ and $\mathcal{P}^+$ are stable by well-sorted substitutions,*
4. *$\mathcal{P}^+$ is stable by conversion,*
5. *$\mathcal{O}^+$ is stable by $\to$-reduction and $\mathcal{O}^-$ is stable by $\to$-equivalence,*
6. *No R-reduction can occur at the root position of terms of $\mathcal{O}\backslash\mathcal{O}^-$,*
7. *Suppose that Γ is s.t. $\mathcal{O}^- \times \mathcal{O}^- \nsubseteq \sim_\Gamma$. If $t \sim_\Gamma u$ with $t \in \mathcal{O}^+$, then there exists a term $u' \in \mathcal{O}^+$ s.t. $u \to_* u'$ and $t \sim_\Gamma u'$. (Resp. if $u \in \mathcal{O}^+$, then there exists a term $t' \in \mathcal{O}^+$ s.t. $t \to_* t'$ and $t' \sim_\Gamma u$.*

4.1 Confluence on well-sorted terms

Lemma 4.2

1. if $t \to t'$ and t is well-sorted (resp. weak), so is t',
2. $\to$ is confluent on well-sorted terms
3. If $\mathrm{Class}(t) \neq \bot$ and $t \to u$, then $\mathrm{Class}(u) = \mathrm{Class}(t)$.

Proof. Property (1) is an immediate induction on the position of the reduction. Property (2) is a immediate consequence of property (1). Property (3) is well-known for $\xrightarrow{\beta\iota R}$ reduction where R is a rewriting system preserving classes. □

4.2 Monotony of conversion

We start with properties of interpretations of algebraised terms. Our first property simply states that if an algebraised term $\mathcal{A}_{\mathcal{R}}(t)(\sigma)$ is evaluated w.r.t. an interpretation $\mathcal{I}$, then for

any relation $\mathcal{R}'$ coarser than $\mathcal{R}$, we can construct an interpretation $\mathcal{I}'$ (independently of t and σ) s.t. $[\![\mathcal{A}_{\mathcal{R}'}(t)(\sigma)]\!]^{\mathcal{I}'}_{\mathcal{M}} = [\![\mathcal{A}_{\mathcal{R}}(t)(\sigma)]\!]^{\mathcal{I}}_{\mathcal{M}}$.

Example 4.3

If $t = u_1 \dot{+} u_2$ with u_1, u_2 non algebraic, then by definition of interpretation:

$$[\![\mathcal{A}_{\mathcal{R}}(t)(\mathbf{nat})]\!]^{\mathcal{I}}_{\mathcal{M}} = +_{\mathcal{M}}(\mathcal{I}(y_1), \mathcal{I}(y_2)) \text{ and } [\![\mathcal{A}_{\mathcal{R}'}(t)(\mathbf{nat})]\!]^{\mathcal{I}'}_{\mathcal{M}} = +_{\mathcal{M}}(\mathcal{I}'(y'_1), \mathcal{I}'(y'_2))$$

where $y_i = \Pi^{\mathbf{nat}}_{\mathcal{R}}(u_i)$ and $y'_i = \Pi^{\mathbf{nat}}_{\mathcal{R}'}(u'_i)$. In order to have the desired equality, if suffices to take $\mathcal{I}'(y'_i) = \mathcal{I}(y_i)$. This definition could be ill-formed if $y'_1 = y'_2$ and $\mathcal{I}(y_1) \neq \mathcal{I}(y_2)$. Assuming $\mathcal{R}'$ coarser than $\mathcal{R}$ eliminates this case. Indeed, having $y'_1 = y'_2$, implies $u_1 \ \mathcal{R}' \ u_2$, which in turns implies $u_1 \ \mathcal{R} \ u_2$ and $y_1 = y_2$.

The lemma we state is a little stronger as it allows the application of a substitution to the interpreted term: if an algebraised term $\mathcal{A}_{\mathcal{R}}(t\theta)(\sigma)$ is evaluated w.r.t. an interpretation $\mathcal{I}$, then for any relation $\mathcal{R}'$ s.t. $\mathcal{R}'\theta \subseteq \mathcal{R}$, we can construct an interpretation $\mathcal{I}'$ (independently of t and σ) s.t. $[\![\mathcal{A}_{\mathcal{R}'}(t)(\sigma)]\!]^{\mathcal{I}'}_{\mathcal{M}} = [\![\mathcal{A}_{\mathcal{R}}(t\theta)(\sigma)]\!]^{\mathcal{I}}_{\mathcal{M}}$.

Example 4.4

If $t = x \dot{+} u$ with u non-algebraic, then

$$[\![\mathcal{A}_{\mathcal{R}}(t\theta)]\!]^{\mathcal{I}}_{\mathcal{M}} = +_{\mathcal{M}}([\![x\theta]\!]^{\mathcal{I}}_{\mathcal{M}}, \mathcal{I}(\Pi^{\mathbf{nat}}_{\mathcal{R}}(u\theta))) \text{ and } [\![\mathcal{A}_{\mathcal{R}'}(t)]\!]^{\mathcal{I}'}_{\mathcal{M}} = +_{\mathcal{M}}(\mathcal{I}'(x^{\mathbf{nat}}), \mathcal{I}'(\Pi^{\mathbf{nat}}_{\mathcal{R}'}(u))).$$

Taking $\mathcal{I}'(x^{\mathbf{nat}}) = [\![x\theta]\!]^{\mathcal{I}}_{\mathcal{M}}$ and $\mathcal{I}'(\Pi_{\mathcal{R}'}(u)) = \mathcal{I}(\Pi_{\mathcal{R}}(u\theta))$ yields the desired result. With respect to previous example, the difference is in the interpretation of variables in the domain of θ.

Let $t = u_1 \dot{+} u_2$ (with u_1, u_2 non-algebraic) as in the previous example, and define $\mathcal{I}'$ with $\mathcal{I}'(\Pi^{\mathbf{nat}}_{\mathcal{R}'}(u_i)) = \mathcal{I}(\Pi^{\mathbf{nat}}_{\mathcal{R}}(u_i\theta))$. Again, this definition is well-formed: if $\Pi^{\mathbf{nat}}_{\mathcal{R}'}(u_1) = \Pi^{\mathbf{nat}}_{\mathcal{R}'}(u_2)$, then $u_1 \ \mathcal{R}' \ u_2$, and thus $u_1\theta \ \mathcal{R} \ u_2\theta$ and $\mathcal{I}(\Pi^{\mathbf{nat}}_{\mathcal{R}}(u_1\theta)) = \mathcal{I}(\Pi^{\mathbf{nat}}_{\mathcal{R}}(u_2\theta))$.

We now state and prove the property:

Lemma 4.5

Let θ a substitution and $\mathcal{R}$, $\mathcal{R}'$ two binary relations on CCIC s.t. $\mathcal{R}' \subseteq \mathcal{R}\theta$. Then, for any $\mathcal{T}$-model $\mathcal{M}$ and $\mathcal{T}$-interpretation $\mathcal{I}$, there exists an interpretation $\mathcal{I}_\theta$ s.t. for any term t and sort $\sigma \in \Lambda_{\mathcal{E}}$:

$$[\![\mathcal{A}_{\mathcal{R}}(t)(\sigma)]\!]^{\mathcal{I}_\theta}_{\mathcal{M}} = [\![\mathcal{A}_{\mathcal{R}'}(t\theta)(\sigma)]\!]^{\mathcal{I}}_{\mathcal{M}}$$

Proof. We define $\mathcal{I}_\theta$ as

$$\begin{aligned}
\mathcal{I}_\theta(x^\sigma) &= [\![\mathcal{A}_{\mathcal{R}'}(x\theta)(\sigma)]\!]^{\mathcal{I}}_{\mathcal{M}} && \text{if } x \in \mathrm{dom}(\theta)\\
\mathcal{I}_\theta(x^\sigma) &= \mathcal{I}(x^\sigma) && \text{if } x \in \mathcal{X}\backslash\,\mathrm{dom}(\theta)\\
\mathcal{I}_\theta(y) &= \mathcal{I}(\Pi^\sigma_{\mathcal{R}'}(u\theta)) && \text{if } y \in \mathcal{Y}^\sigma,\ u \in {\Pi^\sigma_{\mathcal{R}}}^{-1}(y)
\end{aligned}$$

Note that this definition is well defined: if $\Pi_{\mathcal{R}}(u) = \Pi_{\mathcal{R}}(u')$, then $u \ \mathcal{R} \ u'$. Hence, by Lemma hypothesis, $u\theta \ \mathcal{R}' \ u'\theta$ and $\Pi_{\mathcal{R}'}(u\theta) = \Pi_{\mathcal{R}'}(u'\theta)$.

We prove the desired property by an induction on the definition of $\mathcal{A}_{\mathcal{R}}(t)(\sigma)$:

- $t = x \in \mathcal{X}$.

- If $x \in \mathrm{dom}(\theta)$, then $[\![\mathcal{A}_{\mathcal{R}}(x)(\sigma)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}} = \mathcal{I}_\theta(x^\sigma) = [\![\mathcal{A}_{\mathcal{R}'}(x\theta)(\sigma)]\!]^{\mathcal{J}}_{\mathcal{M}}$.
- Otherwise, $[\![\mathcal{A}_{\mathcal{R}}(x)(\sigma)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}} = \mathcal{I}_\theta(x^\sigma) = \mathcal{I}(x^\sigma) = [\![\mathcal{A}_{\mathcal{R}'}(x)(\sigma)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}} = [\![\mathcal{A}_{\mathcal{R}'}(x\theta)(\sigma)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}}$.

- If t is a fully applied term of the form $f\ \vec{T}\ t_1 \dots t_n$ with f of arity $\forall\vec{\alpha}.\tau_1 \times \cdots \times \tau_n \to \tau$ and ξ is a substitution s.t. $\sigma = \tau\xi$, then

$$\begin{aligned}
[\![\mathcal{A}_{\mathcal{R}}(f\ \vec{T}\ t_1 \dots t_n)(\sigma)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}} &= f_{\mathcal{M}}([\![\mathcal{A}_{\mathcal{R}}(t_1)(\tau_1\xi)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}}, \dots, [\![\mathcal{A}_{\mathcal{R}}(t_n)(\tau_n\xi)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}}) \\
&= f_{\mathcal{M}}([\![\mathcal{A}_{\mathcal{R}'}(t_1\theta)(\tau_1\xi)]\!]^{\mathcal{J}}_{\mathcal{M}}, \dots, [\![\mathcal{A}_{\mathcal{R}'}(t_n\theta)(\tau_n\xi)]\!]^{\mathcal{J}}_{\mathcal{M}}) \qquad \text{(by IH)} \\
&= [\![\mathcal{A}_{\mathcal{R}'}(f\ \vec{T\theta}\ t_1\theta \dots t_n\theta)(\tau\xi)]\!]^{\mathcal{J}}_{\mathcal{M}} \\
&= [\![\mathcal{A}_{\mathcal{R}'}(t\theta)(\sigma)]\!]^{\mathcal{J}}_{\mathcal{M}}
\end{aligned}$$

- Otherwise, $[\![\mathcal{A}_{\mathcal{R}}(t)(\sigma)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}} = \mathcal{I}_\theta(\Pi^\sigma_{\mathcal{R}}(t)) = \mathcal{I}(\Pi^\sigma_{\mathcal{R}'}(u\theta))$ with $u \in {\Pi^\sigma_{\mathcal{R}}}^{-1}(y)$ where y denotes $\Pi^\sigma_{\mathcal{R}}(t)$. Now, by definition of u, $t\ \mathcal{R}\ u$, and by assumption, $t\theta\ \mathcal{R}'\ u\theta$. Thus, $\Pi^\sigma_{\mathcal{R}'}(u\theta) = \Pi^\sigma_{\mathcal{R}'}(t\theta)$.

  Since t does not have an algebraic cap and is not a variable, so is $t\theta$. Thus,

$$[\![\mathcal{A}_{\mathcal{R}'}(t\theta)(\sigma)]\!]^{\mathcal{J}}_{\mathcal{M}} = \mathcal{I}(\Pi^\sigma_{\mathcal{R}'}(t\theta)) = [\![\mathcal{A}_{\mathcal{R}}(t)(\sigma)]\!]^{\mathcal{J}_\theta}_{\mathcal{M}}.$$

□

We now carry out *monotony of conversion*.

Notation. For any CCIC binary relation $\mathcal{R}$, we denote by $\mathsf{E}(\mathcal{R})$ the following set of first order equations:

$$\mathsf{E}(\mathcal{R}) = \{\mathcal{A}_{\mathcal{R}}(w_1)(\sigma) = \mathcal{A}_{\mathcal{R}}(w_2)(\sigma) \mid w_1, w_2 \in \mathcal{O}^+, w_1\ \mathcal{R}\ w_2, \sigma \in \Lambda_{\mathcal{E}}\}$$

Lemma 4.6 **Monotony of conversion**

Let $\mathcal{R}$ and $\mathcal{R}'$ be two binary relations on CCIC terms s.t. $\mathcal{R} \subseteq \mathcal{R}'$.

Then for any term t, t' and sort $\sigma \in \Lambda_{\mathcal{E}}$, if $\mathcal{T}, \mathsf{E}(\mathcal{R}) \vDash \mathcal{A}_{\mathcal{R}}(t)(\tau) = \mathcal{A}_{\mathcal{R}}(t')(\tau)$ holds then $\mathcal{T}, \mathsf{E}(\mathcal{R}') \vDash \mathcal{A}_{\mathcal{R}'}(t)(\tau) = \mathcal{A}_{\mathcal{R}'}(t')(\tau)$ holds.

Proof. There exists $\{\mathsf{E}_i\}_{i\in n}$ with $\mathsf{E}_i = (\mathcal{A}_{\mathcal{R}}(w_{1,i})(\tau_i) = \mathcal{A}_{\mathcal{R}}(w_{2,i})(\tau_i)) \in \mathsf{E}(\mathcal{R}))$, s.t.:

$$\mathcal{T} \vDash \mathsf{E}_1 \wedge \cdots \wedge \mathsf{E}_n \Rightarrow \mathcal{A}_{\mathcal{R}}(t)(\sigma) = \mathcal{A}_{\mathcal{R}}(u)(\sigma) \quad (*)$$

We first show that

$$\mathcal{T} \vDash \mathsf{E}'_1 \wedge \ldots \wedge \mathsf{E}'_n \Rightarrow \mathcal{A}_{\mathcal{R}'}(t) = \mathcal{A}_{\mathcal{R}'}(u) \quad (\dagger)$$

where $\mathsf{E}'_i = (\mathcal{A}_{\mathcal{R}'}(w_{1,i})(\tau_i) = \mathcal{A}_{\mathcal{R}'}(w_{2,i})(\tau_i))$.

Let $\mathcal{M}$ be a $\mathcal{T}$-model and $\mathcal{I}$ a $\mathcal{M}$-interpretation. If $[\![\mathsf{E}'_1 \wedge \cdots \wedge \mathsf{E}'_n]\!]^{\mathcal{I}}_{\mathcal{M}} = \bot$ then $(\dagger)$ is satisfied by $\mathcal{I}$. Otherwise, assume that $[\![\mathcal{A}_{\mathcal{R}'}(t)(\sigma)]\!]^{\mathcal{I}}_{\mathcal{M}} \neq [\![\mathcal{A}_{\mathcal{R}'}(u)(\sigma)]\!]^{\mathcal{I}}_{\mathcal{M}}$. We apply Lemma 4.5 (using an empty substitution), and construct a $\mathcal{M}$-interpretation $\mathcal{J}$ s.t.

$$[\![\mathsf{E}_1 \wedge \cdots \wedge \mathsf{E}_n]\!]^{\mathcal{J}}_{\mathcal{M}} = \top \text{ and } [\![\mathcal{A}_{\mathcal{R}}(t)(\sigma)]\!]^{\mathcal{J}}_{\mathcal{M}} \neq [\![\mathcal{A}_{\mathcal{R}}(u)(\sigma)]\!]^{\mathcal{J}}_{\mathcal{M}},$$

contradicting $(*)$.

Therefore, $[\![\mathcal{A}_{\mathcal{R}'}(t)(\sigma)]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![\mathcal{A}_{\mathcal{R}'}(u)(\sigma)]\!]^{\mathcal{I}}_{\mathcal{M}}$, and $[\![F]\!]^{\mathcal{I}}_{\mathcal{M}} = \top$, ending the proof of $(\dagger)$.

Now, by assumption, for all i, $w_{1,i}\ \mathcal{R}'\ w_{2,i}$. Hence, for all i, $\mathsf{E}'_i \in \mathsf{E}(\mathcal{R}')$. Therefore, $\mathcal{T}, \mathsf{E}(\mathcal{R}') \vDash \mathcal{A}_{\mathcal{R}'}(t)(\sigma) = \mathcal{A}_{\mathcal{R}'}(u)(\sigma)$ by $(\dagger)$. □

Monotony of conversion will be used later in several lemmas stating stability of conversion w.r.t. several operations, such as:

Weakening If $\Gamma \subseteq \Delta$, then $\sim_\Gamma \subseteq \sim_\Delta$,

Stability by reduction If $\Gamma \to \Delta$, then $\sim_\Gamma \subseteq \sim_\Delta$,

Stability by substitution If $\theta : \Gamma \hookrightarrow \Delta$, then $\sim_\Gamma \subseteq \sim_\Delta$.

Theses lemmas have all the same proof sketch: an induction on the definition of $\sim_\Gamma$ demonstrating that i) every equation extracted from Γ is extractable from Δ, and that ii) every application of [Ded] of $\sim_\Gamma$ can be translated to an application of [Ded] of $\sim_\Delta$. Monotony of conversion is dealt with by [Ded] case.

4.3 Weakening

We state here the different basic properties of Pure Type Systems. Only the proof of weakening differs from the PTS case.

Lemma 4.7 **Free variables / Subterms / Environments**

1. Let $\Gamma = \overrightarrow{[x_i :^{a_i} T_i]}$. If $\Gamma \vdash t : T$, then
   - $FV(t) \cup FV(T) \subseteq dom(\Gamma)$,
   - for any i, $FV(i) \subseteq \{x_1, \ldots, x_{i-1}\}$.
2. All subterms of a well-formed term are well-formed.
3. Let $\Gamma = \overrightarrow{[x_i :^{a_i} T_i]}$ be a well-formed environment. Then,
   a) if $x_i \in \mathcal{X}^s$, then $[x_1 :^{a_1} T_1], \cdots, [x_{i-1} :^{a_{i-1}} T_{i-1}] \vdash T_i : s_i$,
   b) for any i, $[x_1 :^{a_1} T_1], \cdots, [x_{i-1} :^{a_{i-1}} T_{i-1}] \vdash x_i : T_i$.

Proof. 1. Straightforward induction on $\Gamma \vdash t : T$.

2. Straightforward induction on the typing judgment derivation.

3. Property a) is done by induction on $\Gamma \vdash t : T$. Property b) is an immediate consequence of a) using [Var]. □

We now state two stability lemmas for conversion:

Stability w.r.t. conversion of extractable equations If Γ and Δ are s.t. any extractable equation of Γ is extractable in Δ, then $\sim_\Gamma \subseteq \sim_\Delta$.

Stability by reduction If $\Gamma \to \Delta$, then $\sim_\Gamma \subseteq \sim_\Delta$.

Lemma 4.8 **Conversion weakening**

Let Γ and Δ be two environments s.t. $\forall x \in dom(\Gamma)$, if x is annotated by r in Γ, then i) $x \in dom(\Delta)$, ii) $x\Delta = x\Gamma$, and iii) x is annotated by r in Δ. Then $\sim_\Gamma \subseteq \sim_\Delta$.

Proof. We prove $t \sim_\Delta u$ by induction on the definition of $t \sim_\Gamma u$.

- [Eq]. $\Gamma = \Gamma_1, [x :^r T], \Gamma_2$, $T \to_* w_1 \doteq w_2$ and $w_1, w_2 \in \mathcal{O}^+$

  Then, $\Delta = \Delta_1, [x :^r T], \Delta_2$ and by application of [Eq], $w_1 \sim_\Delta w_2$.

- [Ded]. $\mathcal{T}, E_{\sim_\Gamma} \vDash \mathcal{A}_{\sim_\Gamma}(t)(\sigma) = \mathcal{A}_{\sim_\Gamma}(u)(\sigma)$ with $t, u \in \mathcal{O}^-$, $\sigma \in \Lambda_\mathcal{E}$ and
$$E_{\sim_\Gamma} = \{\mathcal{A}_{\sim_\Gamma}(w_1)(\tau) = \mathcal{A}_{\sim_\Gamma}(w_2)(\tau) \mid w_1, w_2 \in \mathcal{O}^+, \tau \in \Lambda_\mathcal{E}, w_1 \sim_\Gamma w_2\}$$

  Using induction hypothesis and Lemma 4.6, we have
$$\mathcal{T}, E_{\sim_\Delta} \vDash \mathcal{A}_{\sim_\Delta}(t)(\sigma) = \mathcal{A}_{\sim_\Delta}(u)(\sigma)$$
  with
$$E_{\sim_\Delta} = \{\mathcal{A}_{\sim_\Delta}(w_1)(\tau) = \mathcal{A}_{\sim_\Delta}(w_2)(\tau) \mid w_1, w_2 \in \mathcal{O}^+, \tau \in \Lambda_\mathcal{E}, w_1 \sim_\Delta w_2\}.$$
  Thus $t \sim_\Delta u$ by application of [Ded].

- All other cases are done by application of the induction hypothesis. □

Fact 4.9

The conversion relation $\sim_\Gamma \cap \mathcal{O}^+ \cap \mathcal{O}^+$ is transitive and symmetric.

Proof. If $t \sim_\Gamma u \sim_\Gamma v$ with $t, u, v \in \mathcal{O}^+ \subseteq \mathcal{O}^-$, then $t \sim_\Gamma v$ by application of [Ded]. Likewise, if $t \sim_\Gamma u$ with $t, u \in \mathcal{O}^+ \subseteq \mathcal{O}^-$, then $u \sim_\Gamma t$ by application of [Ded]. □

Lemma 4.10 **Stability by reduction**

If $\Gamma \to_* \Delta$, then $\sim_\Gamma \subseteq \sim_\Delta$.

Proof. We prove $t \sim_\Delta u$ by induction on $t \sim_\Gamma u$ and by case on the last rule used.

- [Eq]. $\Gamma = \Gamma_1, [x :^r T], \Gamma_2$, $T \to_* w_1 \doteq w_2$, $w_1, w_2 \in \mathcal{O}^+$

  Suppose that $\Gamma_1, [x :^r T], \Gamma_2 \to_* \Delta_1, [x :^r T'], \Delta_2$ with $T \to_* T'$. Using confluence of $\to$, $T' \to_* w_1' \doteq w_2'$ with $w_1 \to_* w_1'$ and $w_2 \to_* w_2'$. Using stability of $\mathcal{O}^+$ w.r.t. $\to$, $w_1', w_2' \in \mathcal{O}^+$. Thus, $w_1 \to_* w_1' \sim_\Delta w_2' \leftarrow_* w_2$, and $w_1 \sim_\Delta w_2$ by application of [Rw] and Lemma 4.9.

- [Ded]. $\mathcal{T}, E_{\sim_\Gamma} \vDash \mathcal{A}_{\sim_\Gamma}(t)(\sigma) = \mathcal{A}_{\sim_\Gamma}(u)(\sigma)$ with $t, u \in \mathcal{O}^-$, $\sigma \in \Lambda_\mathcal{E}$ and
$$E_{\sim_\Gamma} = \{\mathcal{A}_{\sim_\Gamma}(w_1)(\tau) = \mathcal{A}_{\sim_\Gamma}(w_2)(\tau) \mid w_1, w_2 \in \mathcal{O}^+, \tau \in \Lambda_\mathcal{E}, w_1 \sim_\Gamma w_2\}$$

  As for the weakening proof, using induction hypothesis and Lemma 4.6, we have
$$\mathcal{T}, E_{\sim_\Delta} \vDash \mathcal{A}_{\sim_\Delta}(t)(\sigma) = \mathcal{A}_{\sim_\Delta}(u)(\sigma)$$
  with
$$E_{\sim_\Delta} = \{\mathcal{A}_{\sim_\Delta}(w_1)(\tau) = \mathcal{A}_{\sim_\Delta}(w_2)(\tau) \mid w_1, w_2 \in \mathcal{O}^+, \tau \in \Lambda_\mathcal{E}, w_1 \sim_\Delta w_2\}.$$
  Thus, $t \sim_\Delta u$ by application of [Ded].

- All other cases are done by application of the induction hypothesis. □

We obtain weakening as a immediate consequence of Lemma 4.8.

Lemma 4.11 **Weakening**

Suppose that $\Gamma \vdash t : T$ and $t \sim_\Gamma u$. Then, for any typing environment Δ s.t. $\Gamma \subseteq \Delta$ and $\Delta \vdash$, we have $\Delta \vdash t : T$.

Proof. The proof is as usual by induction on $\Gamma \vdash t : T$, using Lemma 4.8 for [Conv]. □

4.4 Substitutivity

We now come to the stability of substitution for *well-sorted* substitutions.

Lemma 4.12 — **Stability by substitution**

Let $\theta : \Gamma \hookrightarrow \Delta$ and T, T' s.t. $T \sim_\Gamma T'$. Then, $t\theta \sim_\Delta u\theta$.

Proof. By induction on the definition of $T \sim_\Gamma T'$:

- [Rw]. By standard properties of reductions.
- [Eq]. $[x :^r U] \in \Gamma$ with $x \to_* T \doteq T'$ and $T, T' \in \mathcal{O}^+$

  Since $\theta : \Gamma \hookrightarrow \Delta$, we have $x \in \mathrm{dom}(\Delta)$ and $x\Delta = x\Gamma\theta$. Since $T\theta \to_* w_1\theta \doteq w_2\theta$ and $\mathcal{O}^+$ is stable by well-sorted substitutions, then $w_1\theta \sim_\Delta w_2\theta$ using [Eq].
- [Ded]. $E_{\sim_\Gamma} \vDash \mathcal{A}_{\sim_\Gamma}(T)(\sigma) = \mathcal{A}_{\sim_\Gamma}(T')(\sigma)$ with $T, T' \in \mathcal{O}^-$, $\sigma \in \Lambda_\mathcal{E}$ and
  $$E_{\sim_\Gamma} = \{\mathcal{A}_{\sim_\Gamma}(w_1)(\tau) = \mathcal{A}_{\sim_\Gamma}(w_2)(\tau) \mid w_1, w_2 \in \mathcal{O}^+, \tau \in \Lambda_\mathcal{E}, w_1 \sim_\Gamma w_2\}$$

  There exists $\{E_i\}_{i\in n}$ with $E_i = (\mathcal{A}_{\sim_\Gamma}(w_{1,i})(\tau_i) = \mathcal{A}_{\sim_\Gamma}(w_{2,i})(\tau_i)) \in E_{\sim_\Gamma}$ s.t.:
  $$\mathcal{T} \vDash E_1 \wedge \cdots \wedge E_n \Rightarrow \mathcal{A}_{\sim_\Gamma}(T)(\sigma) = \mathcal{A}_{\sim_\Gamma}(T')(\sigma) \quad (*)$$

  By mimicking the proof of monotony of conversion (Lemma 4.6), we can prove:
  $$\mathcal{T} \vDash E_1^\theta \wedge \ldots \wedge E_n^\theta \Rightarrow \mathcal{A}_{\sim_\Delta}(T\theta) = \mathcal{A}_{\sim_\Delta}(T'\theta) \quad (\dagger)$$
  where $E_i^\theta = (\mathcal{A}_{\sim_\Delta}(w_{1,i}\theta)(\tau_i) = \mathcal{A}_{\sim_\Delta}(w_{2,i}\theta)(\tau_i))$.

  Now, by induction hypothesis, for all i, $w_{1,i}\theta \sim_\Delta w_{2,i}\theta$ and by stability of $\mathcal{O}^+$, $w_{1,i}\theta, w_{2,i}\theta \in \mathcal{O}^+$. Hence, for all i, $E_i^\theta \in E_{\sim_\Delta}$, where:
  $$E_{\sim_\Delta} = \{\mathcal{A}_{\sim_\Delta}(w_1)(\tau) = \mathcal{A}_{\sim_\Delta}(w_2)(\tau) \mid w_1, w_2 \in \mathcal{O}^+, \tau \in \Lambda_\mathcal{E}, w_1 \sim_\Delta w_2\}$$
  Therefore, $\mathcal{T}, E_{\sim_\Delta} \vDash \mathcal{A}_{\sim_\Delta}(T\theta)(\sigma) = \mathcal{A}_{\sim_\Delta}(T'\theta)(\sigma)$ by $(\dagger)$. By stability of $\mathcal{O}^-$, $T\theta \in \mathcal{O}^-$ and $T'\theta \in \mathcal{O}^-$. Hence, $T\theta \sim_\Delta T'\theta$ by [Ded].
- Other cases follow by an application of the induction hypothesis, noting that if t is weak, then so is $t\theta$.

□

Corollary 4.13 — **Substitution**

Suppose that $\Gamma \vdash t : T$ and $\theta : \Gamma \rightsquigarrow \Delta$. Then $\Delta \vdash t\theta : T\theta$.

Proof. By induction on the definition of $\Gamma \vdash t : T$:

- [Var] - $t = x \in \mathrm{dom}(\Gamma)$, $T = x\Gamma$. Since $\theta : \Gamma \rightsquigarrow \Delta$, $\Delta \vdash x\theta : x\Gamma\theta$.
- [Conv]. Immediate from induction hypothesis and Lemma 4.12.
- Other cases follow by an application of the induction hypothesis. □

4.5 Product compatibility

Product compatibility is one of the needed properties for proving subject reduction. Indeed, assume that $\Gamma \vdash \lambda[x :^u U'].v\,u : V\{x \mapsto u\}$ with $\Gamma \vdash \lambda[x :^u U'].v : \forall(x :^u U).V$ and $\Gamma \vdash u : U$. For subject reduction to hold, we must have $\Gamma \vdash v\{x \mapsto u\} : V\{x \mapsto u\}$. By inversion of typing, we have $\Gamma, [x :^u U'] \vdash v : V'$ with $\forall(x :^u U').V' \sim^*_\Gamma \forall(x :^u U).V$. We can conclude $\Gamma \vdash v\{x \mapsto u\} : V\{x \mapsto U\}$ only if

$$\forall(x :^a U').V' \sim^*_\Gamma \forall(x :^a U).V \text{ implies } U' \sim^*_\Gamma U \text{ and } V' \sim^*_{\Gamma,[x:^a U']} V.$$

This property is the *product compatibility*. We prove here a stronger property as required by the induction we use later: the *equivalence of non-object caps*.

Definition 4.14

Rules of Figure 4.1 define a family of relation $\{\equiv_\Gamma\}_\Gamma$ on terms of class different from $\bot$, and where each rule whose name is annotated with a $*$ and of the form

$$\frac{E_1 \cdots E_n}{t \equiv_\Gamma u}$$

has to be read as

$$\frac{E_1 \cdots E_n \quad \textit{either } t \textit{ or } u \textit{ is not in } \mathcal{O}^- \quad \mathrm{Class}(t) \neq \bot \quad \mathrm{Class}(u) \neq \bot}{t \equiv_\Gamma u}$$

If $t \equiv_\Gamma u$, we say that t and u have Γ-equivalent non-object caps.

Lemma 4.15

If $t \equiv_\Gamma u$, then $t \sim_\Gamma u$.

Proof. Straightforward induction on the definition of $t \equiv_\Gamma u$. □

Lemma 4.16

Assume that Γ and Δ are environments s.t. $\sim_\Gamma \subseteq \sim_\Delta$. Then $\equiv_\Gamma \subseteq \equiv_\Delta$.

Proof. Straightforward induction on the definition of $t \equiv_\Gamma u$. □

The relation $\equiv_\Gamma$ has been defined s.t. if $t \equiv_\Gamma u$ where t and u are weak terms, then there exists a derivation $t \equiv_\Gamma u$ using only weak terms.

Definition 4.17

We define $\equiv^{\mathcal{W}}_\Gamma$ as $\equiv_\Gamma$ but restricted to weak terms. I.e. $\equiv^{\mathcal{W}}_\Gamma$ is defined by the same rules of $\sim_\Gamma$ where

1. *all occurrences of $\equiv_\Gamma$ are replaced by $\equiv^{\mathcal{W}}_\Gamma$, and*
2. *all terms appearing in the conclusion of a rule are weak.*

Lemma 4.18

If $t, u \in \mathcal{WT}$ and $t \equiv_\Gamma u$, then $t \equiv^{\mathcal{W}}_\Gamma u$.

$$\frac{t \sim_\Gamma u \quad t,u \in \mathcal{O}^-}{t \equiv_\Gamma u}\ [\textsc{Object}] \qquad \frac{s \in \mathcal{S}}{s \equiv_\Gamma s}\ [\textsc{Sort}\ (*)] \qquad \frac{x \in \mathcal{X}^\square}{x \equiv_\Gamma x}\ [\textsc{Var-}\mathcal{X}^\square\ (*)]$$

$$\frac{f \in \underline{\Sigma} \cup \underline{\Lambda} \cup \{\mathrm{Leib}, \doteq\}}{f \equiv_\Gamma f}\ [\textsc{Symb}\ (*)] \qquad \frac{T \equiv_\Gamma U \quad t \equiv_\Gamma u}{\mathrm{Eq}_T(t) \equiv_\Gamma \mathrm{Eq}_T(u)}\ [\textsc{Eq}\ (*)]$$

$$\frac{T \equiv_\Gamma U \quad t \equiv_{\Gamma,[x:^a T]} u}{\lambda[x :^a T].\, t \sim_\Gamma \lambda[x :^a U].\, u}\ [\textsc{Lam}\ (*)] \qquad \frac{T \equiv_\Gamma U \quad t \equiv_{\Gamma,[x:^a T]} u}{\forall(x :^a T).\, t \equiv_\Gamma \forall(x :^a U).\, u}\ [\textsc{Prod}\ (*)]$$

$$\frac{t_1 \equiv_\Gamma u_1 \quad t_2 \equiv_\Gamma u_2 \quad t_1, t_2, u_1, u_2 \text{ weak}}{t_1\, t_2 \equiv_\Gamma u_1\, u_2}\ [\textsc{App}^{\mathcal{W}}\ (*)]$$

$$\frac{t_1 = u_1 \quad t_2 = u_2}{t_1\, t_2 \equiv_\Gamma u_1\, u_2}\ [\textsc{App}^{\mathcal{S}}\ (*)]$$

$$\frac{\begin{array}{c} I \text{ and } I' \text{ are weak inductive types} \\ \mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \text{ and } \mathrm{Elim}(t' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\} \text{ are weak} \\ t = t' \quad I \equiv_\Gamma I' \quad Q \equiv_\Gamma Q' \quad \vec{v} \equiv_\Gamma \vec{v}' \quad \vec{f} \equiv_\Gamma \vec{f}' \end{array}}{\mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \equiv_\Gamma \mathrm{Elim}(t' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\}}\ [\textsc{Elim}^{\mathcal{W}}\ (*)]$$

$$\frac{t = t' \quad I = I' \quad Q = Q' \quad \vec{v} = \vec{v}' \quad \vec{f} = \vec{f}'}{\mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \equiv_\Gamma \mathrm{Elim}(t' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\}}\ [\textsc{Elim}^{\mathcal{S}}\ (*)]$$

$$\frac{A \equiv_\Gamma A' \quad \overrightarrow{C(X)} \equiv_\Gamma \overrightarrow{C'(X)}}{\mathrm{Ind}(X : A)\{\overrightarrow{C(X)}\} \equiv_\Gamma \mathrm{Ind}(X : A')\{\overrightarrow{C'(X)}\}}\ [\textsc{Ind}\ (*)] \qquad \frac{I \equiv_\Gamma I'}{I^{[n]} \equiv_\Gamma I'^{[n]}}\ [\textsc{Constr}\ (*)]$$

Figure 4.1: Non-object cap equivalence

Proof. Straightforward induction on $t \equiv_\Gamma u$, using the fact that subterms of weak terms are weak terms. □

Note 8

The rules $[\textsc{App}^{\mathcal{S}}]$ and $[\textsc{Elim}^{\mathcal{S}}]$ of $\equiv_\Gamma^{\mathcal{W}}$ are subsumed by $[\textsc{App}^{\mathcal{W}}]$ and $[\textsc{Elim}^{\mathcal{W}}]$. Thus, we can remove them from the definition of $\equiv_\Gamma^{\mathcal{W}}$.

Lemma 4.19 **Substitutivity for $\equiv_\Gamma$**

If $\theta : \Gamma \hookrightarrow \Delta$ and $t \equiv_\Gamma u$, then $t\theta \equiv_\Delta u\theta$.

Proof. Straightforward induction on $t \equiv_\Delta u$, using the substitutivity of $\sim_\Gamma$. □

We now come to the main property of weak terms: the stability of conversion by convertible substitutions.

Lemma 4.20

Suppose that $t \sim_\Gamma u$ with $t, u \in \mathcal{WT}$ and $\theta, \theta' : \Gamma \hookrightarrow \Delta$ s.t.

- $\mathrm{dom}(\theta) = \mathrm{dom}(\theta')$,
- for all $x \in \mathrm{dom}(\theta)$, $x\theta \sim_\Delta x\theta'$,

Then $t\theta \sim^*_\Delta t\theta'$.

Proof. By the substitutivity lemma, $t\theta \sim_\Delta u\theta$. We now prove by induction on the structure of u that if u is weak, then for any environment Ω, $u\theta \sim_{\Delta,\Omega} u\theta'$.

- [Var] - $t = x$. If x is not in $\mathrm{dom}(\theta)$, this is immediate. Otherwise, $x\theta \sim_\Delta x\theta'$ by assumption, hence $x\theta \sim_{\Delta,\Omega} x\theta'$ by weakening.
- [App] - $t = t_1\, t_2$. By induction hypothesis, $t_1\theta \sim_{\Delta,\Omega} t_1\theta'$ and $t_2\theta \sim_{\Delta,\Omega} t_2\theta'$. Since co-domains of θ and θ' are weak, $t_1\theta, t_2\theta, t_1\theta'$ and $t_2\theta'$ are all weak. We conclude by application of $[\textsc{App}^{\mathcal{W}}]$.
- [Elim] - $t = \mathrm{Elim}(v : I\,[\vec{u}] \to Q)\{\vec{f}\}$. We first apply our induction hypothesis on I, $\vec{u}$, Q and $\vec{f}$. Since t is weak, $v\theta = v = v\theta'$. Since co-domains of θ and θ' are weak, $t\theta$ and $t\theta'$ are weak terms. We conclude by applying rule $[\textsc{Elim}^{\mathcal{W}}]$.
- All other cases are done by a straightforward application of the induction hypothesis. □

Lemma 4.21

Suppose that $t \equiv_\Gamma u$ with $t, u \in \mathcal{WT}$ and $\theta, \theta' : \Gamma \hookrightarrow \Delta$ s.t.

- $\mathrm{dom}(\theta) = \mathrm{dom}(\theta')$,
- for all $x \in \mathrm{dom}(\theta)$, $x\theta \sim_\Delta x\theta'$,

Then $t\theta \equiv_\Delta u\theta'$.

Proof. By induction on the definition of $t \equiv^{\mathcal{W}}_\Gamma u$, t, u being weak terms:

- [Object]. This is a consequence of Lemmas 4.20.
- $[\textsc{App}^{\mathcal{W}}]$. $t = t_1\, t_2$, $u = u_1\, u_2$, $t_i \equiv_\Gamma u_i$, t_1, t_2, u_1, u_2 are weak

  By application of the induction hypothesis, $t_i\theta \equiv_\Delta u_i\theta'$. Moreover, by assumption on the co-domain of θ and θ', $t_1\theta, t_2\theta, u_1\theta'$ and $u_2\theta'$ are all weak. Hence, by application of the $[\textsc{App}^{\mathcal{W}}]$ rule, $t\theta \equiv_\Delta u\theta'$.
- $[\textsc{Elim}^{\mathcal{W}}]$. $t = \mathrm{Elim}(v : I\,[\vec{v}] \to Q)\{\vec{f}\}$, $u = \mathrm{Elim}(v' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\}$

  Since t and u are weak, $v\theta = v = v' = v'\theta$. Moreover, by assumption on θ and θ', $t\theta$ and $u\theta'$ are weak. We conclude by application of induction hypothesis and of rule $[\textsc{Elim}^{\mathcal{W}}]$.
- All other cases are done by a straightforward application of induction the hypothesis, using that if v is weak then $v\theta$ is weak. □

We need a last technical lemma about $\equiv_\Gamma$-conversion and $\to$-reduction:

Lemma 4.22

Suppose that $t \equiv_\Gamma u$ and $t \to t'$, where $t, u \in \mathcal{WT}$. Then, there exists a term u' s.t. $t' \equiv_\Gamma u'$ and $u \to_\leqslant u'$.

Proof. By induction on $t \equiv_\Gamma^{\mathcal{W}} u$, t and u being weak.

- [Object]. Since $t, u \in \mathcal{O}^-$ and $\mathcal{O}^-$ is stable by reduction, we have $t' \in \mathcal{O}^-$. Hence, $t' \sim_\Gamma t \sim_\Gamma u$. By Lemma 4.9, $t' \sim_\Gamma u$. Hence, $t' \equiv_\Gamma u$.
- [App$^{\mathcal{W}}$]. $t = t_1\, t_2$, $u = u_1\, u_2$, $t_i \equiv_\Gamma u_i$

  If reduction occurs in t_1 or t_2, we conclude by an application of the induction hypothesis. Otherwise, $\to$-reduction is necessarily a β or R-reduction. Since $t \notin \mathcal{O}^-$, by assumption on $\mathcal{O}\backslash\mathcal{O}^-$, this cannot be an R-reduction. Thus, $t_1 = \lambda[x :^u T].\, v$ and $t' = v\{x \mapsto t_2\}$. By inversion of $t_1 \approx_\Gamma u_1$, we have $u_1 = \lambda[x :^u U].\, w$ with $T \equiv_\Gamma U$ and $v \equiv_{\Gamma,[x:^u T]} w$. Thus, $u \to w\{x \mapsto u_2\}$. Since T is annotated with the unrestricted annotation, we have $v \equiv_\Gamma w$. From $v \equiv_\Gamma w$, $t_2 \equiv_\Gamma u_2$ and $t_2, u_2 \in \mathcal{WT}$, we obtain $v\{x \mapsto t_2\} \equiv_\Gamma w\{x \mapsto u_2\}$ by Lemma 4.21.
- [App^{S}]. We simply take $u' = t'$.
- [Elim$^{\mathcal{W}}$]. $t = \mathrm{Elim}(v : I\,[\vec{w}] \to Q)\{\vec{f}\}$, $u = \mathrm{Elim}(v' : I'\,[\vec{w}'] \to Q')\{\vec{f}'\}$, $v = v'$

  If reduction does not occur at the root position, we conclude by application of the induction hypothesis. Otherwise, the reduction must be a ι-reduction.

  Then, $v = v' = T^{[k]}\ \vec{p}$, $t \xrightarrow{\iota} \Delta[I, X, C_k, f_k, Q, \vec{f}, \vec{p}]$ and $I = \mathrm{Ind}(X : \forall(\overrightarrow{x : A}).\, s)\{\vec{C_i}\}$.

  Moreover, $u \xrightarrow{\iota} \Delta[I', X, C'_k, f'_k, Q', \vec{f}', \vec{p})]$ with $C_k \equiv_\Gamma C'_k$.

  We now prove by induction on the constructor type $C(X)$ that if $C(X) \equiv_\Gamma C'(X)$ and f, f' are two weak terms at type level s.t. $f \equiv_\Gamma f'$ and $\vec{q}$ is a vector a closed terms, then $\Delta[I, X, C(X), f, Q, \vec{f}, \vec{q}] \equiv_\Gamma \Delta[I', X, C'(X), f, Q, \vec{f}', \vec{q}]$:

  - If $C(X) = X\,\vec{m}$, then by inversion, $C'(X) = X\,\vec{m}'$, and:

    $$\Delta[I, X, C(X), f, Q, \vec{f}, \vec{q}] = f \equiv_\Gamma f' = \Delta[I', X, C'(X), f', Q, \vec{f}', \vec{q}].$$

  - If $C(X) = \forall(x : B).\, D$ where X does not occur in B and $\vec{q} = r\,\vec{r}$, then by inversion $C'(X) = \forall(x : B').\, D'$ with $B \equiv_\Gamma B'$ and $D \equiv_{\Gamma,[x:B]} D'$. Hence:

    $$\Delta[I, X, C(X), f, Q, \vec{f}, \vec{q}] = \Delta[I, X, D\{x \mapsto r\}, f\,r, Q, \vec{f}, \vec{r}]$$
    $$\Delta[I', X, C(X), f', Q', \vec{f}', \vec{q}] = \Delta[I', X, D'\{x \mapsto r\}, f'\,r, Q', \vec{f}', \vec{r}]$$

    Since f and f' are weak and r is closed, $f\,r$ and $f'\,r$ are weak. If $f\,r$ or $f'\,r$ are not in $\mathcal{O}^-$, then from $f \equiv_\Gamma f'$, we have $f\,r \equiv_\Gamma f'\,r$. Otherwise, $f \sim_\Gamma f'$ and thus $f\,r \sim_\Gamma f'\,r$ using [App$^{\mathcal{W}}$]. Hence, $f\,r \equiv_\Gamma f'\,r$ using [Object].

    Moreover, since $x \notin \mathcal{X}^-_\star \cup \mathcal{X}^-_\square$ (by assumption on the formation of constructor types) and r is weak (since r is closed), $\{x \mapsto r\}$ is a well-sorted substitution and $D\{x \mapsto r\} \equiv_\Gamma D'\{x \mapsto r\}$ by substitutivity (Lemma 4.19).

    Hence, by application of the induction hypothesis:

    $$\Delta[I, X, D\{x \mapsto r\}, f\,r, Q, \vec{f}, \vec{q}] \equiv_\Gamma \Delta[I', X, D'\{x \mapsto r\}, f'\,r, Q', \vec{f}', \vec{q}]$$

  - If $C(X) = \forall(x : B).\, D$ with X occurring in B and $q = r\,\vec{r}$, then, since I is weak, B is of the form $X\,\vec{w}$. By inversion of $C(X) \equiv_\Gamma C'(X)$, $C'(X) = X\,\vec{w}'$ with $\vec{w} \equiv_\Gamma \vec{w}'$. Hence:

$$\Delta[I, X, C(X), f, Q, \vec{f}, \vec{q}] =$$
$$\Delta[I, X, D\{x \mapsto r\}, f\, r\ \mathrm{Elim}(r : I\,[Q] \to \vec{w})\{\vec{f}\}, Q, \vec{f}, \vec{r}]$$
$$\Delta[I', X, C(X), f', Q', \vec{f}', \vec{q}] =$$
$$\Delta[I', X, D'\{x \mapsto r\}, f'\, r\ \mathrm{Elim}(r : I'\,[Q'] \to \vec{w}')\{\vec{f}'\}, Q', \vec{f}', \vec{r}]$$

As for previous case, one can check that $f\, r\ \mathrm{Elim}(r : I\,[Q] \to \vec{w})\{\vec{f}\}$ and $f'\, r\ \mathrm{Elim}(r : I'\,[Q'] \to \vec{w}')\{\vec{f}'\}$ are $\equiv_\Gamma$-convertible. Hence, by application of the induction hypothesis:

$$\Delta[I, X, C(X), f, Q, \vec{f}, \vec{q}] \equiv_\Gamma \Delta[I', X, C(X), f', Q', \vec{f}', \vec{q}]$$

· All other cases use a straightforward application of the induction hypothesis. □

We can now state and prove the *preservation of non-object cap* Lemma:

Lemma 4.23

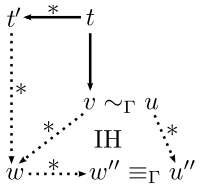
If $t \sim_\Gamma u$ and $t \to_* t'$ then there exists t'' and u'' s.t. $t' \to_* t''$, $u \to_* u''$ and $t'' \equiv_\Gamma u''$.

Proof. By induction on the definition of $t \sim_\Gamma u$. If t, u are in $\mathcal{O}^-$, then so is t'. Thus, $t' \sim_\Gamma t \sim_\Gamma u$ and $t' \equiv_\Gamma u$ by [Object]. We now consider rules which do not operate at $\mathcal{O}^-$.

Note that if $t \notin \mathcal{O}^-$, then for any t' s.t. $t \to_* t'$, $t' \notin \mathcal{O}^-$.

· [Refl]. Immediate since $t = u$.

· [Rw-Left]. $t \to v$ and $v \sim_\Gamma u$

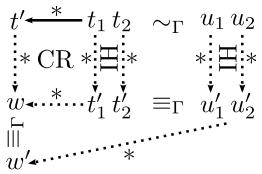


If $t \to_* t'$, then by confluence of $\to$, there exists a term w s.t. $t' \to_* w$ and $v \to_* w$. By induction hypothesis on $v \sim_\Gamma u$, there exists two terms w'' and u'' s.t. $w \to_* w''$, $u \to_* u''$ and $w'' \equiv_\Gamma u''$.

· [Rw-Right]. $u \to u'$ and $t \sim_\Gamma u'$

If $t \to_* t'$, by induction hypothesis on $t \sim_\Gamma u'$, there exists t'' and u'' s.t. $t \to_* t''$, $u \to u' \to_* u''$ and $t'' \equiv_\Gamma u''$.

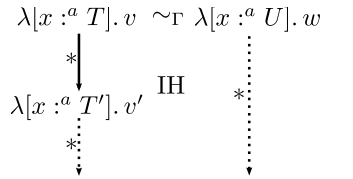
· [App$^{\mathcal{W}}$]. $t = t_1\, t_2$, $u = u_1\, u_2$, $t_i \sim_\Gamma u_i$ and t_1, t_2, u_1, u_2 are weak



By application of the induction hypothesis to $t_1 \sim_\Gamma u_1$ and $t_2 \sim_\Gamma u_2$, there exists t'_1, t'_2, u'_1, u'_2 s.t. $t_i \to_* t'_i$, $u_i \to_* u'_i$ and $t'_i \equiv_\Gamma u'_i$. Note that t'_i, u'_i are weak since weak terms are closed under reduction.

Suppose now that $t \to_* t'$. By confluence, there exists a term w s.t. $t'_1\, t'_2 \to_* w$ and $t' \to_* w$. $t'_1\, t'_2$ and $u'_1\, u'_2$ being weak as application of weak terms, we obtain $t'_1\, t'_2 \equiv_\Gamma u'_1\, u'_2$ by application of [App$^{\mathcal{W}}$]. By repeated application of Lemma 4.22, there exists a term w' s.t. $u'_1\, u'_2 \to_* w'$ and $w \equiv_\Gamma w'$.

· [Lam]. $t = \lambda[x :^a T].v$, $u = \lambda[x :^a U].w$, $T \sim_\Gamma U$, $v \sim_{\Gamma,[x:^a T]} w$



Since no reductions can occur at the root position of t, $t \to_* \lambda[x :^a T'].v'$ with $T \to_* T'$ and $v \to_* v'$. Applying the induction hypothesis to $v \sim_{\Gamma,[x:^a T]} w$ and $T \sim_\Gamma U$, there exist v'', T'', w'', U'' s.t. $T' \to_* T'' \equiv_\Gamma U'' \leftarrow_* U$ and $v' \to_* v'' \equiv_{\Gamma,[x:^a T]} w'' \leftarrow_* w$. By Lemma 4.10 and 4.16, $v'' \equiv_{\Gamma,[x:^a T'']} w''$. We conclude $\lambda[x :^a T''].v'' \equiv_\Gamma \lambda[x :^a U''].w''$ by application of [Lam].

- [Prod]. As for the [Lam] case.
- [Elim]. $t = \mathrm{Elim}(v : I\,[\vec{w}] \to Q)\{\vec{f}\}$, $u = \mathrm{Elim}(v' : I'\,[\vec{w}'] \to Q')\{\vec{f}'\}$, $v \leftrightarrow_* v'$

  This case is similar to the [App] one. Using induction hypothesis and confluence of $\to$, there exist w_1 and w_2 s.t. $t \to_* w_1$, $u \to_* w_2$ and $w_1 \equiv_\Gamma w_2$.

  If $t \to_* t'$, then by confluence of $\to$, there exists a term t'' s.t. $t' \to_* t''$ and $w_1 \to_* t''$. By repeated application of Lemma 4.22, we obtain that there exists a term u'' s.t. $w_2 \to_* u''$ and $t'' \equiv_\Gamma u''$.
- The cases [Ind], [Constr], [Eq] and [Symb] are done by application of the induction hypothesis. □

A first consequence of preservation of non-object cap is the stability of conversion by environment conversion:

Definition 4.24 — **Environment conversion**

We define the relation $\sim$ on typing environments as the smallest relation s.t.

$$\text{if } \Gamma = \Gamma_1, [x :^a T], \Gamma_2 \text{ and } \Delta = \Gamma_1, [x :^a T'], \Gamma_2 \text{ with } T \sim_{\Gamma_1} T', \text{ then } \Gamma \sim \Delta.$$

Lemma 4.25

If $\Gamma \sim \Delta$, then $\sim_\Gamma = \sim_\Delta$.

Proof. Let $\Gamma = \Gamma_1, [x :^a T], \Gamma_2$ and $\Delta = \Gamma_1, [x :^a T'], \Gamma_2$ with $T \sim_{\Gamma_1} T'$. Assume that $t \sim_\Gamma u$. We prove $t \sim_\Delta u$ by induction on the definition of $t \sim_\Gamma u$:

- [Eq]. $[y :^r U] \in \Gamma$, $y\Gamma \to_* t \doteq u$, $t, u \in \mathcal{O}^+$

  If $x \neq y$, this is immediate since $x\Gamma = x\Delta$ and x is annotated by r in Γ.

  Otherwise, $U \sim_\Gamma y\Delta$. From Lemma 4.23, there exist t', u', t'', u'' s.t. $t \to_* t'$, $u \to_* u'$ and $y\Delta \to_* t'' \doteq u''$ with $t' \sim_{\Gamma_1} t''$, $u' \sim_{\Gamma_1} u''$ and $t', u' \in \mathcal{O}^+$. Using stability of $\mathcal{O}^+$, there exist $t^3, u^3 \in \mathcal{O}^+$ s.t. $t'' \to_* t^3$, $u'' \to_* u^3$, $t' \sim_{\Gamma_1} t^3$ and $u' \sim_{\Gamma_1} u^3$. Hence, $t^3 \sim_\Delta u^3$ using [Eq]. Using weakening and multiple application of the [Rw] rule, $t \sim_\Delta t^3$ and $u \sim_\Delta u^3$. Using transitivity and symmetry on $\mathcal{O}^+$ (Lemma 4.9) of $\sim_\Delta$, $t \sim_\Delta u$.
- [Ded]. As for the [Ded] case of the Lemma 4.10.
- All other cases are done by a straightforward application of the induction hypothesis □

Lemma 4.26 — **Product compatibility**

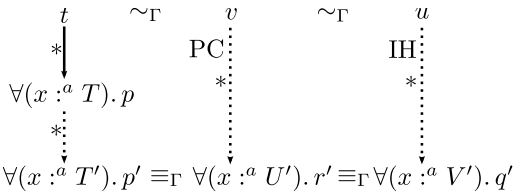
1. If $\forall(x :^a T).t \sim^*_\Gamma \forall(x :^a U).u$, then $T \sim^*_\Gamma U$ and $t \sim^*_{\Gamma,[x:^a T]} u$.

2. If $s_1 \sim^*_\Gamma s_2$, then $s_1 = s_2$.

Proof. 1. We do an induction on the length of the conversion $t \sim^*_\Gamma u$, proving that if $t \to_* \forall(x :^a T).\, t'$, then $u \to_* \forall(x :^a U).\, u'$ with $T \sim^*_\Gamma U$ and $t' \sim^*_{\Gamma,[x:^a T]} u'$.

Assume that $t \sim_\Gamma v \sim^*_\Gamma u$ and $t \to_* \forall(x :^a T).\, p$. Using Lemma 4.23, there exist T', p', V', q' s.t. $t \to_* \forall(x :^a T').\, p'$, $v \to_* \forall(x :^a V').\, q'$ with $T' \equiv_\Gamma V'$ and $p' \equiv_{\Gamma,[x:^a T']} q'$. Since $\equiv_\Gamma \subseteq \sim^*_\Gamma$, $T' \sim^*_\Gamma V'$ and $p' \sim^*_{\Gamma,[x:^a T']} q'$.

By application of the induction hypothesis, there exists U', r' s.t. $u \to_* \forall(x :^a U').\, r'$ with $V' \sim^*_\Gamma U'$ and $q' \sim^*_{\Gamma,[x:^a V']} r'$. From $T \to_* T' \sim^*_\Gamma V'$ and by Lemma 4.25, we have $p \to_* p' \sim^*_{\Gamma,[x:^a T]} q' \sim^*_{\Gamma,[x:^a T]} r'$. Hence, $T \sim^*_\Gamma U'$ and $p \sim^*_\Gamma r'$.



2. As for the [Prod] case. □

4.6 Correctness of types

Lemma 4.27

1. If $\Gamma \vdash \forall(x :^a T).\, t : s$, then $\Gamma, [x :^a T] \vdash t : s$.
2. If $\Gamma \vdash \mathrm{Ind}(X : A)\{\overrightarrow{C_i(X)}\} : T$, then $A \sim^*_\Gamma T$, $\Gamma \vdash T : \square$ and $\Gamma, [X : A] \vdash C_i(X)$ for any i.

Proof. 1. The judgment $\Gamma \vdash \forall(x :^a T).\, t$ ends by an application of [Prod] followed by several applications of [Weak] or [Conv]. Thus, we have $\Gamma, [x :^a T] \vdash t : s'$ with $s' = s_0 \sim_{\Gamma_0} s_1 \sim_{\Gamma_1} \cdots s_n \sim_{\Gamma_n} s_n = s$ with $\Gamma_0 \subseteq \Gamma_1 \subseteq \cdots \subseteq \Gamma_n = \Gamma$. I.e., $s' = s_1 = \cdots = s_n = s$.

2. The judgment $\Gamma \vdash \mathrm{Ind}(X : A)\{\overrightarrow{C_i(X)}\} : T$ ends by an application of [Ind] followed by several applications of [Weak] or [Conv]. We use the same argument as before:

   - $A = T_0 \sim_{\Gamma_0} T_1 \sim_{\Gamma_1} \cdots T_n \sim_{\Gamma_n} T$ with $\Gamma_0 \subseteq \Gamma_1 \subseteq \cdots \subseteq \Gamma_n = \Gamma$
   - $\Gamma_0 \vdash A : \square$ and $\forall i, \Gamma_0, [X : A] \vdash C_i(X) : \star$.

We obtain the desired result using weakening of typing and monotony of conversion. □

Lemma 4.28 — **Preservation of classes**

If $t \sim_\Gamma u$, then $\mathrm{Class}(t) = \mathrm{Class}(u)$.

Proof. Straightforward induction on $t \sim_\Gamma u$, noticing that [Eq] and [Ded] constrain the converted terms to be in $\mathcal{O}$. □

Lemma 4.29 — **Correction of types**

If $\Gamma \vdash t : T$ then

1. $T = \square$ or $\Gamma \vdash T : s$ with $s \in \mathcal{S}$,
2. $\mathrm{Class}(T) \neq \bot$ and $\mathrm{Class}(t) \neq \bot$, and
3. $\mathrm{Class}(T) = \mathrm{Class}(t) + 1$.

Proof. The proof is as usual, we detail the changing cases. We do an induction on the definition of $\Gamma \vdash t : T$.

- [App]. $t = u\,v$, $\Gamma \vdash u : \forall (x :^u V).W$, $\Gamma \vdash v : V$ and $T = W\{x \mapsto v\}$

  By induction hypothesis, $\Gamma \vdash \forall (x :^u V).W : s$ and from Lemma 4.27, $\Gamma, [x :^u T] \vdash W : s$.

  Using side conditions of [App], we know that $v \in \mathcal{WT}$ and that if $x \in \mathcal{X}^-_\star$ (resp. $x \in \mathcal{X}^+_\square$), then $v \in \mathcal{O}^+$ (resp. $v \in \mathcal{P}^+$). Moreover, by induction hypothesis on $\Gamma \vdash t : \forall (x :^a U).V$, we know that $\mathrm{Class}(x) = \mathrm{Class}(U)$. Thus, $\{x \mapsto v\} : \Gamma, [x :^u T] \rightsquigarrow \Gamma$ and by substitutivity Lemma, we obtain $\Gamma \vdash W\{x \mapsto t\} : s$.

- [Constr]. $t = I^{[n]}$, $T = C_n(I)$ with $I = \mathrm{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$

  Then, from Lemma 4.27, we have $A \sim^*_\Gamma T$, $\Gamma \vdash T : \square$ and $\Gamma, [X :^u A] \vdash C_j(X) : \star$. Form Lemma 4.7 and definition of well-sorted terms, $X \in \mathcal{X}^\square - \mathcal{X}^-_\square$. By application of the induction hypothesis, $\mathrm{Class}(T) = \mathcal{K}$. Hence, by preservation of classes by conversion, $\mathrm{Class}(A) = \mathcal{K}$. By a second application of the induction hypothesis, $\mathrm{Class}(I) = \mathrm{Class}(T) - 1 = \mathcal{P}$. Hence, $\mathrm{Class}(X) = \mathrm{Class}(I)$. Since X is annotated with u and $\mathrm{Class}(X) = \mathrm{Class}(I) = \mathcal{P}$, we have $\{X \mapsto I\} : \Gamma, [X : A] \rightsquigarrow \Gamma$, and we can apply the substitutivity Lemma to $\Gamma, [X :^u A] \vdash C_j(X)$. Hence, $\Gamma \vdash C_n(I) : \star$.

- All other cases are immediate or is a straightforward consequence of the induction hypothesis. □

4.7 Subject reduction

Lemma 4.30 — **Subject reduction for $\mathtt{R}$ ([4])**

If $\Gamma \vdash t : T$ and $t \xrightarrow{\mathtt{R}} t'$, then $\Gamma \vdash t' : T$.

Proof. Following the proof of [4]. □

Lemma 4.31 — **Subject reduction**

If $\Gamma \vdash t : T$ and $t \to_* t'$, $\Gamma \vdash t' : T$.

Proof. As usual, we prove by induction on the definition of $\Gamma \vdash t : T$ that the two following properties hold:

- if $\Gamma \to_\leqslant \Delta$, then $\Delta \vdash t : T$,
- if $t \to_\leqslant t'$, then $\Gamma \vdash t' : T$.

Proof is standard here, and we only detail new cases:

- [Var] - $t = x \in \mathcal{X}$.

Reduction is in Γ. If the reduction is not in the type associated to x in Γ, the result is immediate. Otherwise, $\Gamma = \Gamma_1, [x :^a T], \Gamma_2$, $\Gamma' = \Gamma_1, [x :^a T'], \Gamma_2$ with $T \to_{\leqslant} T'$ and there must be a sub-derivation $\Gamma_1 \vdash T : s$ in $\Gamma \vdash t : T$. Thus, by weakening $\Gamma' \vdash T : s$, and since $\Gamma' \vdash x : T'$ and $T \to_{\leqslant} T'$, by conversion, $\Gamma' \vdash x : T$.

- [App] - $t = u\,v$, $\Gamma \vdash u : \forall(x :^a V).W$, $\Gamma \vdash v : V$, $T = W\{x \mapsto v\}$.

  If the reduction occurs in u, this is immediate. If $v \xrightarrow{\beta} v'$ then by induction hypothesis, we can easily derive that $\Gamma' \vdash u\,v' : W\{x \mapsto v'\}$ and $\Gamma' \vdash u\,v : W\{x \mapsto v\}$. Since $W\{x \mapsto v\} \xrightarrow{\beta}_* W\{x \mapsto v'\}$, we have $W\{x \mapsto v\} \sim_\Gamma W\{x \mapsto v'\}$ and, $\Gamma' : u\,v' : W\{x \mapsto v\}$ by conversion.

  Otherwise, we have either a β or $\mathcal{R}$-reduction at root position.

  - If a β-reduction occurs at the root of t, then $u = \lambda[x :^a V'].w$ and $t \xrightarrow{\beta} w\{x \mapsto v\}$. By inversion, $\Gamma, [x :^a V'] \vdash w : W'$ with $\forall(x :^a V).W \sim_\Gamma \forall(x :^a V').W'$ and $a = u$. By type structure compatibility, $V \sim^*_\Gamma V'$ and $W \sim^*_{\Gamma,[x:^a V']} W'$. Thus, by conversion, $\Gamma, [x :^a V'] \vdash w : W$ and $\Gamma \vdash v : V'$.

    We can then apply the substitution lemma: $\Gamma \vdash w\{x \mapsto v\} : W\{x \mapsto v\}$.
  - If a $\mathcal{R}$-reduction occurs at the root of t, we apply Lemma 4.30.

- [Elim]. Identical to the proof of subject reduction of CIC [48].

- All other cases are immediate. □

4.8 Type unicity

Lemma 4.32 — **Type unicity**

Assume that $\sim_\Gamma$ is an equivalence relation on well-formed terms. If $\Gamma \vdash t : T_1$ and $\Gamma \vdash t : T_2$ then $T_1 \sim_\Gamma T_2$.

Proof. The proof is as usual by case on the head structure of t, and then by inversion, using type structure compatibility. □

4.9 Strong normalization

In 2001, Blanqui [3] defined the Calculus of Algebraic Constructions (CAC), an extension of the Calculus of Constructions whose conversion relation is the union of β-reduction and an arbitrary rewriting system $\mathcal{R}$. Combination of $\xrightarrow{\beta}$-reduction with rewriting has been studied since the end of the 80's, starting with the work of Tannen [44] for the confluence of the combination of $\xrightarrow{\beta}$ and first order rewriting, and next the works of Tannen and Gallier [45, 46], and of Okada [34] for the strong normalization and confluence of the combination of $\xrightarrow{\beta}$ and polymorphic first-order rewriting system. In CAC, $\mathcal{R}$ can be an arbitrary higher-order rewriting system including type level rewriting as long as $\mathcal{R}$ verifies the so-called *general schema*, which is a generalization of primitive recursion at higher types introduced by Jouannaud and Okada [29] and further generalized by Blanqui [3, 4].

A concise and elegant proof of strong normalization of $\xrightarrow{\beta\mathcal{R}}$ for well formed terms can be found in [4]. CAC can capture numerous higher-order systems, including CIC, for

which the rules are simply beta, weak-iota and strong-iota. See [5] for a description of the embedding of CIC in CAC.

Our proof of strong normalization shall indeed mimic Blanqui's proof of strong normalization for CAC. We shall recall the main lemmas, and carry out in detail the proofs which are not verbatim copies of Blanqui's proofs.

In [2], Barthe proved that strong normalization is compatible with *proof-irrelevance*, that equates all object-level expressions of a given type. Because his result is restricted to PTSs, it does not apply to CCIC. Our result can therefore be seen as a generalization of Barthe's one. Here, we build a proof-irrelevant interpretation for the Calculus of Inductive Constructions, which equates object level terms as in Barthe.

A modified version of the Calculus of Algebraic Construction

We give here the definition of the Calculus of Algebraic Construction with a slight modification of its conversion relation. In the initial definition CAC, terms are equal if and only if they are convertible w.r.t. a fixed rewriting relation. In CAC$^+$, two terms are convertible is they $\Rightarrow$-reduce, for a given rewriting system $\Rightarrow$, to two terms having the same non-object level cap. Assuming $\Rightarrow$ confluent, we have a strict extension of CAC, since our conversion relation clearly captures $\Rightarrow$-convertibility.

Note that we are not interested in the logical consistency of CAC$^+$, nor its decidability. We deliberately define a coarse conversion relation so that the meta-theory of CAC$^+$ is straightforward, but not too coarse so that $\Rightarrow$ is strongly normalizing in CAC$^+$.

Definition 4.33 — **Terms of CAC$^+$**

Let Ψ be a set of function symbols. We denote by $\Psi^\star$ (resp. $\Psi^\Box$) the set of object level function symbols *(resp.* type level function symbols*). The algebra of* pseudo-terms *of CAC$^+$ is defined by:*

$$t, u, T, U, \ldots := f \in \Psi \mid s \in \mathcal{S} \mid x \in \mathcal{X} \mid \forall(x : T).\, t \mid \lambda[x : T].\, t \mid t\, u$$

We here too give a layered definition of pseudo-terms of CAC$^+$:

Definition 4.34 — **Syntactic classes**

The pairwise disjoint syntactic classes of CAC$^+$ called objects *($\mathcal{O}$),* predicates *($\mathcal{P}$),* kinds *($\mathcal{K}$), $\Box$ are defined in Figure 4.2.*

$$\begin{aligned}
\mathcal{O} &::= \mathcal{X}^\star \mid f \in \Psi^\star \mid \mathcal{O}\,\mathcal{O} \mid \mathcal{O}\,\mathcal{P} \mid \lambda[x^\star :^a \mathcal{P}].\, \mathcal{O} \mid \lambda[x^\Box :^a \mathcal{K}].\, \mathcal{O} \\
\mathcal{P} &::= \mathcal{X}^\Box \mid f \in \Psi^\Box \mid \mathcal{P}\,\mathcal{O} \mid \mathcal{P}\,\mathcal{P} \mid \lambda[x^\star :^a \mathcal{P}].\, \mathcal{P} \mid \lambda[x^\Box :^a \mathcal{K}].\, \mathcal{P} \\
&::= \forall(x^\star :^a \mathcal{P}).\, \mathcal{P} \mid \forall(x^\Box :^a \mathcal{K}).\, \mathcal{P} \\
\mathcal{K} &::= \star \mid \forall(x^\star :^a \mathcal{P}).\, \mathcal{K} \mid \forall(x^\Box :^a \mathcal{K}).\, \mathcal{K} \\
\Box &::= \Box
\end{aligned}$$

Figure 4.2: CAC$^+$ terms classes

As for CCIC, this enumeration defined a post-fixed successors function $+1$ on classes and we define $\mathrm{Class}(t) = \mathcal{D}$ if $t \in \mathcal{D}$ and $\mathcal{D} \in \{\mathcal{O}, \mathcal{P}, \mathcal{K}, \square\}$, and $\mathrm{Class}(t) = \bot$ otherwise.

Translation of CCIC to CAC$^+$

Definition 4.35 — **Translation of CCIC to CAC$^+$ (Definition 7.3 of [5]**

We define the translation of CCIC well-formed terms to CAC$^+$, written $\langle t \rangle$, by induction on the definition of $\Gamma \vdash t : T$:

$$
\begin{aligned}
\langle x \rangle &= x \\
\langle f \rangle &= \overline{f} && f \in \underline{\Sigma} \cup \underline{\Lambda} \cup \{\doteq, \mathrm{Leib}\} \\
\langle \mathrm{Eq}_T(t) \rangle &= \overline{\mathrm{Eq}}\,\langle T \rangle \langle t \rangle \\
\langle t\,u \rangle &= \langle t \rangle \langle u \rangle \\
\langle \lambda[x :^a T].\, t \rangle &= \lambda[x : \langle T \rangle].\, \langle t \rangle \\
\langle \forall(x :^a T).\, t \rangle &= \lambda[x : \langle T \rangle].\, \langle t \rangle \\
\langle I \rangle &= f_I && I = \mathrm{Ind}(X : A)\{\overrightarrow{C(X)}\} \\
\langle \mathrm{Elim}(t : I\,[\vec{u}] \to Q)\{\vec{f}\} \rangle &= \mathrm{SElim}_I^Q\,\langle \vec{u} \rangle \langle t \rangle \langle \vec{f} \rangle && \text{if } Q \text{ is of the form } \forall(\overrightarrow{x : A})(y : I\,\vec{x}).\, K \\
\langle \mathrm{Elim}(t : I\,[\vec{u}] \to Q)\{\vec{f}\} \rangle &= \mathrm{WElim}_I\,\langle Q \rangle \langle \vec{u} \rangle \langle t \rangle \langle \vec{f} \rangle && \text{otherwise} \\
\langle I^{[i]} \rangle &= f_i^I
\end{aligned}
$$

where $\overline{f}$, $\overline{\mathrm{Eq}}$, f_I, SElim_I^Q, WElim_I and f_i^I are symbols of Ψ.

Having this translation, we can now transpose the notion of *weak terms* to CAC$^+$:

Definition 4.36 — **CAC$^+$ weak terms**

A term $t \in$ CAC$^+$ is weak *if there exists $u \in$ CCIC s.t. u is well-formed in CCIC and $\langle u \rangle = t$.*

Typing judgments

Assume that, for any symbol f of Ψ, we attach a type $\tau_f \in \mathrm{CAC}^+$.

Definition 4.37 — **Conversion relation**

Rules of Figures 4.3 define a binary relation $\cong$ on terms of class different from $\bot$, and where each rule whose name is annotated with a $*$ and of the form

$$\frac{E_1 \cdots E_n}{t \equiv_\Gamma u}$$

has to be read as

$$\frac{E_1 \cdots E_n \quad t, u \notin \mathcal{O} \quad \mathrm{Class}(t) \neq \bot \quad \mathrm{Class}(u) \neq \bot}{t \equiv_\Gamma u}$$

We then define the CAC$^+$ conversion relation $\simeq_\Gamma$ by $t \simeq_\Gamma u$ if there exists $t', u' \in$ CAC$^+$ s.t. $t \Rightarrow_* t'$, $u \Rightarrow_* u'$ and $t' \cong u'$.

$$\frac{t, u \in \mathcal{O}}{t \approx u}\ [\textsc{Object}] \qquad \frac{t_1 \approx u_1 \quad t_2 \approx u_2 \quad t_1, u_1, t_2, u_2 \text{ are weak}}{t_1\, t_2 \approx u_1\, u_2}\ [\textsc{App}^{\mathcal{W}}\ (*)]$$

$$\frac{t_1 = u_1 \quad t_2 = u_2}{t_1\, t_2 \approx u_1\, u_2}\ [\textsc{App}^{\mathcal{S}}\ (*)]$$

$$\frac{s \in \mathcal{S}}{s \approx s}\ [\textsc{Sort}\ (*)] \qquad \frac{x \in \mathcal{X}^{\square}}{x \approx x}\ [\textsc{Var-}\mathcal{X}^{\square}\ (*)] \qquad \frac{f \in \Psi}{f \approx f}\ [\Psi\ (*)]$$

$$\frac{T \approx_\Gamma U \quad t \approx u}{\lambda[x : T].\, t \approx \lambda[x : U].\, u}\ [\textsc{Lam}\ (*)] \qquad \frac{T \approx U \quad t \approx u}{\forall(x : T).\, t \approx \forall(x : U).\, u}\ [\textsc{Prod}\ (*)]$$

Figure 4.3: CAC$^+$ Conversion Relation

Definition 4.38 **CAC$^+$ typing judgement**

The typing judgment $\Gamma \Vdash t : T$, for CAC$^+$, is defined by the rules of Figure 4.4.

$$\frac{}{\Vdash \star : \square}\ [\textsc{Ax-1}] \qquad \frac{\Vdash \tau_f : s \quad f \in \Psi}{\Vdash f : \tau_f}\ [\textsc{Symb}]$$

$$\frac{\Gamma \Vdash T : s_T \quad \Gamma, [x : T] \Vdash U : s_U}{\Gamma \Vdash \forall(x : T).\, U : s_U}\ [\textsc{Prod}]$$

$$\frac{\Gamma \Vdash \forall(x : T).\, U : s \quad \Gamma, [x : T] \Vdash u : U}{\Gamma \Vdash \lambda[x : T].\, u : \forall(x : T).\, U}\ [\textsc{Lam}]$$

$$\frac{\Gamma \Vdash V : s \quad \Gamma \Vdash t : T \quad s \in \{\star, \square\} \quad x \in \mathcal{X}^s \setminus \mathrm{dom}(\Gamma)}{\Gamma, [x : V] \Vdash t : T}\ [\textsc{Weak}]$$

$$\frac{x \in \mathrm{dom}(\Gamma) \cap \mathcal{X}^{s_x} \quad \Gamma \Vdash x\Gamma : s_x}{\Gamma \Vdash x : x\Gamma}\ [\textsc{Var}]$$

$$\frac{\Gamma \Vdash t : T \quad \Gamma \Vdash T : s \quad \Gamma \Vdash T' : s' \quad T \simeq_\Gamma T'}{\Gamma \Vdash t : T'}\ [\textsc{Conv}]$$

Figure 4.4: CAC$^+$ Typing Rules

As for CCIC, we define a notion of well-formed substitution:

Definition 4.39

Let Γ and Δ two CAC$^+$ typing environments and θ a CAC$^+$ substitution. θ is well-formed from Γ to Δ, written $\theta : \Gamma \rightsquigarrow \Gamma$ if for all $x \in \mathrm{dom}(\theta)$, $\Delta \Vdash x\theta : x\Gamma\theta$.

Correctness of conversion

Lemma 4.40 **Embedding of conversion**

1. The relation $\approx$ is reflexive.
2. Assume that T and T' are two CCIC well-formed terms s.t. $T \equiv_\Gamma T'$. Then, $\langle T\rangle \approx \langle T'\rangle$.

Proof. 1. Straightforward induction on the definition of $\approx$.

2. Straightforward induction on the definition of $T \equiv_\Gamma T'$.

□

Lemma 4.41 **Correctness of translation (Theorem 7.1 of [5])**

There exists a rewriting relation $\Rightarrow$ and function symbol types τ_f s.t.

1. If $\Gamma \vdash t : T$, then $\langle\Gamma\rangle \Vdash \langle t\rangle : \langle T\rangle$.
2. If $\Gamma \vdash t : T$ and $t \to t'$, then $\langle t\rangle \Rightarrow \langle t'\rangle$.

Proof. Following the proof of Theorem 7.1 of [5], only the [Conv] case changes. Assume that $\Gamma \vdash t : T'$ is derived using the [Conv] rule from $\Gamma \vdash t : T$, $\Gamma \vdash T : s$, $\Gamma \vdash T' : s'$ and $T \sim_\Gamma T'$.

By Lemma 4.23, there exists two CCIC terms U and U' s.t. $T \to_* U$, $T' \to_* U'$ and $U \equiv_\Gamma U'$. By application of the induction hypothesis, $\langle T\rangle \Rightarrow_* \langle U\rangle$ and $\langle T'\rangle \Rightarrow_* \langle U'\rangle$. By Lemma 4.40, U and U' being well-formed under Γ by subject reduction, $\langle U\rangle \approx \langle U'\rangle$. Hence $\langle T\rangle \simeq \langle T'\rangle$.

By application of the induction hypothesis to all the premises and application of the [Conv] rule, $\langle\Gamma\rangle \Vdash \langle t\rangle : \langle T'\rangle$. □

Meta-theory of CAC$^+$

Before proving strong normalization of $\Rightarrow$ for well-formed terms of CAC$^+$, we must prove some basic meta-theoretical properties, namely substitutivity, product compatibility and correctness of types.

Lemma 4.42

If $\theta : \Gamma \rightsquigarrow \Delta$ and $T \simeq U$, then $T\theta \simeq U\theta$.

Proof. Assume that $T \Rightarrow_* T' \approx U' \Leftarrow_* U$. Then, $T\theta \Rightarrow_* T'\theta$ and $U\theta \Rightarrow_* U'\theta$ from properties of rewriting. A straightforward induction on the definition of $U \approx U'$ gives $U\theta \approx U'\theta$. □

Corollary 4.43 **Substitutivity**

If $\theta : \Gamma \rightsquigarrow \Delta$ and $\Gamma \Vdash t : T$, then $\Delta \Vdash t\theta : T\theta$.

Lemma 4.44

1. If $\forall(x : U).\, V \approx \forall(x : U').\, V'$, then $U \approx U'$ and $V \approx V'$.
2. If $s_1 \approx s_2$, then $s_1 = s_2$.

Proof. Immediate using inversion of $\forall(x:U).V \approx \forall(x:U').V'$ and $s_1 \approx s_2$. □

Corollary 4.45 — **Pre-product compatibility**

1. If $\forall(x:U).V \simeq \forall(x:U').V'$, then $U \simeq U'$ and $V \simeq V'$.
2. If $s_1 \simeq s_2$, then $s_1 = s_2$.

Proof.

1. If $\forall(x:U).V \simeq \forall(x:U').V'$, then

$$\forall(x:U).V \Rightarrow_* \forall(x:U_1).V_1 \text{ and } \forall(x:U').V' \Rightarrow_* \forall(x:U_2).V_2$$

   with $\forall(x:U_1).V_1 \approx \forall(x:U_2).V_2$. By Lemma 4.44, $U_1 \approx U_2$ and $V_1 \approx V_2$. Hence, $U \simeq U'$ and $V \simeq V'$.

2. If $s_1 \simeq s_2$, then $s_1 \approx s_2$, s_1, s_2 being $\Rightarrow$-normal. Hence, $s_1 = s_2$ by Lemma 4.44.

□

Lemma 4.46

If $T \approx U$ and $T \Rightarrow T'$, then there exists U' s.t. $U \Rightarrow_{\leqslant} U'$ and $T' \approx'_U$.

Proof. Similar to the proof of Lemma 4.22. □

Corollary 4.47 — **Product compatibility**

1. If $\forall(x:U).V \simeq^* \forall(x:U').V'$, then $U \simeq^* U'$ and $V \simeq^* V'$.
2. If $s_1 \simeq^* s_2$, then $s_1 = s_2$.

Proof. Following the proof of 4.26. □

Lemma 4.48

If $t \simeq u$ with $\mathrm{Class}(t) \neq \bot$ and $\mathrm{Class}(u) \neq \bot$, then $\mathrm{Class}(t) = \mathrm{Class}(u)$.

Proof. Straightforward induction on the definition of $t' \approx u'$, where t' and u' are s.t. $t \Rightarrow_* t'$ and $u \Rightarrow_* u'$. □

Lemma 4.49 — **Correctness of types**

If $\Gamma \Vdash t : T$ then

1. $T = \square$ or $\Gamma \Vdash T : s$ with $s \in \mathcal{S}$,
2. $\mathrm{Class}(T) \neq \bot$ and $\mathrm{Class}(t) \neq \bot$, and
3. $\mathrm{Class}(T) = \mathrm{Class}(t) + 1$.

Proof. The proof is identical to the one of CAC, using classes preservation of $\simeq$. □

Strong normalization in CAC^+

Definition 4.50 — **Neutral terms**

A CAC^+ term is neutral *if it is not of the form $\lambda[x:T].t$, or a not fully applied term $f\vec{u}$.*

Notation. We write $\mathcal{SN}$ (resp. $\mathcal{WN}$, $\mathcal{NT}$) for the set of $\Rightarrow$-strongly normalizing CAC^+ terms (resp. the set of $\Rightarrow$-weakly normalizing CAC^+ terms, the set of neutral terms).

Definition 4.51 — **Reducibility candidates (Definition 32 of [4])**

We inductively define the set $\mathcal{R}_t$ of the interpretations for the terms of type t in CAC^+, the ordering $\leqslant_t$ on $\mathcal{R}_t$, an element $\top_t \in \mathcal{R}_t$, and an internal operation $\bigwedge_t$ on the powerset of $\mathcal{R}_t$ as follows:

- *if $t \neq \square$ and $\Gamma \vdash t : \square$, then:*
$$\mathcal{R}_t = \{\varnothing\},\ \leqslant_t = \subseteq,\ \top_t = \varnothing,\ \textstyle\bigwedge_t(\mathfrak{R}) = \top_t.$$
- *$\mathcal{R}_s$ is the set of all the subsets R of CAC^+ s.t.:*
  - **(R1)** $R \subseteq \mathcal{SN}$
  - **(R2)** *If $t \in R$ and $t \Rightarrow t'$, then $t' \in R$*
  - **(R3)** *If $t \in \mathcal{NT}$ and $t \Rightarrow t'$ implies $t' \in R$, then $t \in R$*

  Furthermore, $\leqslant_s = \subseteq$, $\top_s = \mathcal{SN}$, $\bigwedge_s(\mathfrak{R}) = \bigcap(\mathfrak{R})$ (using $\top_s$ as neutral element for $\cap$).
- *$\mathcal{R}_{\forall(x:U).K}$ is the set of functions R from $CAC^+ \times \mathcal{R}_U$ to $\mathcal{R}_K$ s.t. $R(u,S) = R(u',S)$ whenever $u \Rightarrow u'$. Furthermore:*
  - $\top_{\forall(x:U).K}(u,S) = \top_K$,
  - $\bigwedge_{\forall(x:u).K}(\mathfrak{R})(u,S) = \bigwedge_K(\{R(u,S) \mid R \in \mathfrak{R}\})$,
  - $R \leqslant_{\forall(x:U).K} R'$ *if and only if for all* (u,S), $R(u,s) \leqslant_K R'(u,S)$.

Definition 4.52 — **Interpretation schema (Definition 37 of [4])**

A candidate assignment *is a function ξ from $\mathcal{X}$ to $\bigcup_{t\in\mathcal{L}} \mathcal{R}_t$. A candidate assignment ξ,* validates *an environment Γ, written $\Gamma \vDash \Gamma$, if for all $x \in \mathrm{dom}\,\Gamma$, $x\xi \in \mathcal{R}_{x\Gamma}$. An interpretation of a symbol f is an element of $\mathcal{R}_{\tau_f}$. An* interpretation *of a set $\mathcal{G}$ of symbols is a function I which, to each symbol $g \in \mathcal{G}$, associate a interpretation I_g of g.*

The interpretation *of t w.r.t. candidate assignment ξ, an interpretation I and a substitution θ, is defined by induction on the structure of t as follows:*

$$\begin{aligned}
[\![t]\!]^I_{\xi,\theta} &= \top_t \text{ if } t \in \mathcal{O} \cup \mathcal{S}\\
[\![f]\!]^I_{\xi,\theta} &= I_f\\
[\![x]\!]^I_{\xi,\theta} &= x\xi\\
[\![\forall(x:U).V]\!]^I_{\xi,\theta} &= \{t \in \mathcal{L} \mid \forall u \in [\![U]\!]^I_{\xi,\theta}, \forall S \in \mathcal{R}_U, t\,u \in [\![V]\!]^I_{\xi^S_x,\theta^u_x}\}\\
[\![\lambda[x:U].v]\!]^I_{\xi,\theta}(u,S) &= [\![v]\!]^I_{\xi^S_x,\theta^u_x}\\
[\![t\,u]\!]^I_{\xi,\theta} &= [\![t]\!]^I_{\xi,\theta}(u\theta, [\![u]\!]^I_{\xi,\theta})
\end{aligned}$$

where $\theta^u_x = \theta \cup \{x \mapsto u\}$ and $\xi^S_x = \xi \cup \{x \mapsto S\}$.

A substitution is adapted *to a Γ-assignment ξ if $\mathrm{dom}(\theta) \subseteq \mathrm{dom}(\Gamma)$ and, for all $x \in \mathrm{dom}(\theta)$, $x\theta \in [\![x\Gamma]\!]^I_{\xi,\theta}$. A pair (ξ,θ) is Γ-valid, written $\xi,\theta \vDash \Gamma$, if $\xi \vDash \Gamma$ and θ is adapted to ξ.*

Lemma 4.53 **Invariance by reduction (Lemma 65 of [4])**

If $\Gamma \vdash t : T$, $t \to t'$, $\xi \vDash \Gamma$ and $t\theta \in \mathcal{WN}$, then $[\![t]\!]_{\xi,\theta} = [\![t']\!]_{\xi,\theta}$

Proof. Following the proof of Lemma 65 of [4]. □

Lemma 4.54 **Invariance by conversion on non-object cap**

Assume that $T \approx U$. Then $[\![T]\!]_{\xi,\theta} = [\![U]\!]_{\xi,\theta}$.

Proof. Straightforward induction on the definition of $T \approx U$. □

Lemma 4.55 **Computability of symbols**

There exists an interpretation of Ψ s.t. for all $f \in \Psi$, $f \in [\![\tau_f]\!]$.

Proof. Following the proof of Lemmas 63 and 68 of [4]. □

Lemma 4.56 **Computability of well-formed terms (Lemma 69 of [4])**

If $\Gamma \Vdash t : T$ and $\xi, \theta \vDash \Gamma$ then $t\theta \in [\![T]\!]_{\xi,\theta}$.

Proof. Following the proof of Lemma 69 of [4]. The proof is done by induction on the definition of $\Gamma \Vdash t : T$. Only the case [Conv] is modified.

We have then $\Gamma \vdash t : U$, $\Gamma \vdash U : s_U$, $\Gamma \vdash T : s_T$ and $U \simeq_\Gamma T$. By application of the induction hypothesis, $t\theta \in [\![U]\!]_{\xi,\theta}$, $U\theta \in [\![s_U]\!]_{\xi,\theta} = \top_{s_u} = \mathcal{SN}$ and $T\theta \in [\![s_T]\!]_{\xi,\theta} = \top_{s_T} = \mathcal{SN}$.

From definition of $\simeq_\Gamma$, there exists two CAC$^+$ terms U', T' s.t. $T \Rightarrow_* T'$, $U \Rightarrow_* U'$ $T' \approx U'$.

By Lemma 4.53, $[\![T]\!]_{\xi,\theta} = [\![T']\!]_{\xi,\theta}$ and $[\![U]\!]_{\xi,\theta} = [\![U']\!]_{\xi,\theta}$. By Lemma 4.54, $[\![T']\!]_{\xi,\theta} = [\![U']\!]_{\xi,\theta}$. Hence, $[\![U]\!]_{\xi,\theta} = [\![T]\!]_{\xi,\theta}$ and $t\theta \in [\![T]\!]_{\xi,\theta}$. □

Theorem 4.1 **Strong normalization of $\Rightarrow$**

Assume that $\Gamma \Vdash t : T$. Then, t is strongly normalizing.

Proof. Let $x\xi = \top_{x\Gamma}$ for all $x \in \mathrm{dom}(\Gamma)$. Since $\xi \vDash \Gamma$ and the identity substitution ι is adapted to ξ, $t \in S = [\![T]\!]_{\xi,\iota}$. Now, either $T = \square$ or $\Gamma \Vdash T : s$ for some $s \in \mathcal{S}$. If $T = \square$, then $S = \top_\square = \mathcal{SN}$. If $\Gamma \Vdash T : s$, then $T \in \mathcal{R}_s$ and $\mathcal{R}_s \subseteq \mathcal{SN}$ by **(R1)**. □

Corollary 4.57

Every typable term of CCIC is $\to$-strongly normalizing.

Proof. Assume that $\Gamma \vdash t : T$. By Lemma 4.41, $\langle\Gamma\rangle \Vdash \langle t\rangle : \langle T\rangle$. By Lemma 4.1, $\langle t\rangle$ is $\Rightarrow$-strongly normalizing. By Lemma 4.41, t is necessary $\to$-strongly normalizing. □

CHAPTER 5

Deciding CCIC

We now move to the decidability of the type checking of CCIC, that is, given a valid environment Γ, a type T such that $\Gamma \vdash T : s \in \mathcal{S}$, and an arbitrary term t, checking whether $\Gamma \vdash t : T$. We do not decide the calculus in all its generality but instead take an instance of CCIC:

- We take Presburger arithmetic as the embedded logic $\mathcal{T}$. See Section 5.3 for a discussion about deciding more theories.
- The set of extractable terms $\mathcal{O}^+$ is defined as the set of pure algebraic terms,
- The set of convertible terms $\mathcal{O}^-$ is defined as the set of all object-level terms but terms of the form $x\,\vec{t}$ with $x \in \mathcal{X}^\star \backslash \mathcal{X}^-_\star$ or terms of the form $f\,\vec{t}$ not fully applied terms.
- The set of extractable type variables $\mathcal{P}^+$ contain all the terms which are convertible, in some environment, to **nat**.
- We of course assume that the rewriting system $\rightarrow$ respects the conditions of Blanqui [4], which implies its strong normalization on well-typed terms.

Definition 5.1 — **Extractable terms**

The set $\mathcal{O}^+$ of extractable terms is made of all the terms which are pure algebraic (i.e. of the form: $t ::= \mathbf{0} \mid \mathbf{S}\,t \mid t \dot{+} t \mid x \in \mathcal{X}^-_\star$). We denote by $\mathcal{A}$ the set of pure algebraic terms.

The set $\mathcal{X}^-_\square$ being of no use here (we do not have parametric sorts), we take $\mathcal{X}^-_\square = \varnothing$. Moreover, having only one sort, we take $\mathcal{Z}_{\mathbf{nat}} = \mathcal{X}^-_\star$ ($\mathcal{Z}_{\mathbf{nat}}$ was defined as the set of variables used for the abstraction, w.r.t. the sort **nat**, of the variables of $\mathcal{X}$), and we define $x^{\mathbf{nat}} = x$ for $x \in \mathcal{X}$. The set $\mathcal{Y}_{\mathbf{nat}}$ of abstraction variables for alien of sort **nat** is denoted by $\mathcal{Y}$. (All these sets where defined in Definition 3.47 and above)

Hence, the algebraisation of a pure algebraic term t result in the currification of all the symbols of t. From now on, we will write t for the algebraisation $\mathcal{A}_{\mathcal{R}}(t)$ of the pure algebraic term t.

Note that in our case, the [Ded] rule of $\sim_\Gamma$ is now:

$$\frac{E \vDash \mathcal{A}_{\sim_\Gamma}(t) = \mathcal{A}_{\sim_\Gamma}(u) \quad t,u \in \mathcal{O}^- \qquad E = \{w_1 = w_2 \mid w_1 \sim_\Gamma w_2, w_1, w_2 \in \mathcal{O}^+\}}{t \sim_\Gamma u}\ [\textsc{Ded}]$$

Type-checking is decomposed as follow. The rules of Figure 3.5 and 3.6 providing no algorithm (due to the presence of the conversion rule), a new syntax oriented typing judgment $\vdash_i$ is defined such that:

$$\Gamma \vdash t : T \text{ if and only if } \Gamma \vdash_i t : T' \text{ with } T \sim_\Gamma T'.$$

Described in Section 5.2, this transformation is classical and done as usual by integrating the conversion rule of $\vdash$ in the application rule [12]. Decidability of type-checking in CCIC is then reduced to the decidability of the conversion relation $\sim_\Gamma$. Deciding $\sim_\Gamma$ is carried out in Section 5.1.

5.1 Decidability of conversion relation

The main idea, again, is to eliminate the non-structural rules, here rules [Rw-Left] and [Rw-Right] of Figure 3.7, and therefore to introduce a weaker conversion relation $\approx_\Gamma$ defined by structural rules only. As a result, the conversion relationship will be related by the property:

$$\sim_\Gamma = \xrightarrow{!} \approx_\Gamma \xleftarrow{!}.$$

Since conversion for non-weak terms was restricted to be $\leftrightarrow_*$, their $\to$-normal forms become syntactically equal. As a consequence, the rules of $\sim_\Gamma$ which differ from those of $\approx_\Gamma$ are: i) [Refl], now restricted to constants and variables. ii) [Eq], now internalized in [Ded]. iii) [Elim$^{\mathcal{W}}$], for which conversion for the first argument is now syntactic equality, and iv) [Ded] which now works with algebraic equations which can be directly computed from $\Gamma_\downarrow$.

Besides, most rules check whether the assumptions made in the environments are consistent with the theory $\mathcal{T}$. This check is absolutely crucial to guarantee the termination of the algorithm. Note that, in the case of an implementation, this check is only needed at the beginning and when traversing a binder, resulting in an increase of the set of extracted equations.

Weak conversion $\approx_\Gamma$

Definition 5.2 — **Environment equations**

For any typing environment Γ, we define the set of Γ-equations as

$$\mathrm{Eq}(\Gamma) = \{t = u \mid [x :^r T] \in \Gamma, x\Gamma \to_* t \doteq u, t, u \in \mathcal{O}^+\}$$

Note that, for any $[x :^r T] \in \Gamma$, there is a unique equation $t \doteq u$ $(t, u \in \mathcal{O}^+)$ s.t. $T \to_* t \doteq u$ as terms in $\mathcal{O}^+$ are normal and $\to$ is confluent.

Definition 5.3 — **Weak conversion relation $\approx_\Gamma$**

Rules of Figures 5.1 and 5.2 define a family of relation $\{\approx_\Gamma\}_\Gamma$.

Example 5.4

Let $\Gamma = [c : \mathbf{nat}], [p :^r (\lambda[x : \mathbf{nat}].\, x)\,\mathbf{0} \doteq c]$. The only extractable equation of Γ being $p\Gamma_\downarrow$, $\mathrm{Eq}(\Gamma) = \{0 = c\}$. We clearly have $(\lambda[x : \mathbf{nat}].\, x \dot{+} x)\,\mathbf{0} \sim_\Gamma c$. Indeed, by [Eq], $\mathbf{0} \sim_\Gamma c$. Hence $\mathbf{0} \dot{+} \mathbf{0} \sim_\Gamma c$ by [Ded] and $(\lambda[x : \mathbf{nat}].\, x \dot{+} x)\,\mathbf{0} \sim_\Gamma c$ by [Rw].

On the contrary, $(\lambda[x : \mathbf{nat}].\, x + x)\,\mathbf{0} \approx_\Gamma c$ does not hold, $(\lambda[x : \mathbf{nat}].\, x + x)\,\mathbf{0}$ not being $\approx_\Gamma$-convertible to its reduct. As expected, the result is recovered by first normalizing the terms being converted. Here, (if we assume $\mathcal{R}$ to be empty):

$$\frac{}{\star \approx_\Gamma \star}\ [\textsc{Refl-}\star] \qquad \frac{}{\square \approx_\Gamma \square}\ [\textsc{Refl-}\square]$$

$$\frac{\mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot \text{ or } x \notin \mathcal{O}^-}{x \approx_\Gamma x}\ [\textsc{Refl-}\mathcal{X}] \qquad \frac{\begin{array}{c}\mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot \text{ or } f \notin \mathcal{O}^- \\ f \in \underline{\Sigma} \cup \underline{\Lambda} \cup \{\mathrm{Leib}, \doteq\}\end{array}}{f \approx_\Gamma f}\ [\textsc{Symb}]$$

$$\frac{\begin{array}{c}\mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot \\ T \approx_\Gamma U \quad t \approx_\Gamma u\end{array}}{\mathrm{Eq}_T(t) \approx_\Gamma \mathrm{Eq}_U(u)}\ [\textsc{Eq}] \qquad \frac{t, u \in \mathcal{O}^- \quad \mathcal{T}, \mathrm{Eq}(\Gamma) \models \bot}{t \approx_\Gamma u}\ [\textsc{Unsat}]$$

$$\frac{\begin{array}{c}T \approx_\Gamma U \quad t \approx_{\Gamma, [x :^a T]} u \\ \mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot \text{ or } \lambda[x :^a T].\, t \text{ and } \lambda[x :^a U].\, u \text{ not in } \mathcal{O}^-\end{array}}{\lambda[x :^a T].\, t \approx_\Gamma \lambda[x :^a U].\, u}\ [\textsc{Lam}]$$

$$\frac{T \approx_\Gamma U \quad t \approx_{\Gamma, [x :^a T]} u}{\forall (x :^a T).\, t \approx_\Gamma \forall (x :^a U).\, u}\ [\textsc{Prod}]$$

$$\frac{\begin{array}{c}t_1 \equiv u_1 \quad t_2 \equiv u_2 \\ \mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot \text{ or } t_1\, t_2 \text{ and } u_1\, u_2 \text{ not in } \mathcal{O}^- \\ t_1\, t_2 \text{ or/and } u_1\, u_2 \text{ is not weak}\end{array}}{t_1\, t_2 \approx_\Gamma u_1\, u_2}\ [\textsc{App}^{\mathcal{S}}]$$

$$\frac{\begin{array}{c}t_1 \approx_\Gamma u_1 \quad t_2 \approx_\Gamma u_2 \quad t_i, u_i \text{ are weak} \\ \mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot \text{ or } t_1\, t_2 \text{ and } u_1\, u_2 \text{ not in } \mathcal{O}^-\end{array}}{t_1\, t_2 \approx_\Gamma u_1\, u_2}\ [\textsc{App}^{\mathcal{W}}]$$

$$\frac{\begin{array}{c}\mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot \quad t = C_t[a_1, \ldots, a_k] \quad u = C_u[a_{k+1}, \ldots, a_{k+l}] \\ C_t \text{ or } C_u \text{ is a non-empty algebraic context} \\ \text{all the } a_i\text{'s have empty algebraic caps} \\ \text{the } c_i\text{'s are fresh variables of } \mathcal{Y} \text{ s.t. } c_i = c_j \text{ iff } a_i \approx_\Gamma b_j \\ \mathcal{T}, \mathrm{Eq}(\Gamma) \models C_t[c_1, \ldots, c_k] = C_u[c_{k+1}, \ldots, c_{k+l}]\end{array}}{t \approx_\Gamma u}\ [\textsc{Ded}]$$

Figure 5.1: Conversion relation $\approx_\Gamma^?$ (Part. 1)

$$((\lambda[x : \mathbf{nat}].\, x \dot{+} x)\, \mathbf{0})_\downarrow = \mathbf{0} \dot{+} \mathbf{0} \approx_\Gamma c$$

We show in next section that this result always holds.

Decidability of $\approx_\Gamma$

Fact 5.5

1. Assume that E is a set of $\mathcal{T}$-equations s.t. $\mathcal{T} \models E$ and let $t \in \mathcal{O}^+$. For any term $u \in$ CCIC s.t. $\mathcal{T}, E \models t = \mathcal{A}_{\mathcal{R}}(u)$, we have $u \in \mathcal{O}^+$.
2. If $\mathcal{T}, E \models t = u$, then $\mathcal{T}, E \models t_{\downarrow_R} = u_{\downarrow_R}$.

$$\frac{\begin{array}{c} t = t' \quad I \approx_\Gamma I' \quad Q \approx_\Gamma Q' \quad \vec{v} \approx_\Gamma \vec{v}' \quad \vec{f} \approx_\Gamma \vec{f}' \\ \mathcal{T}, \mathrm{Eq}(\Gamma) \not\models 0 = 1 \text{ or } \mathrm{Elim}(t,\ldots)\{\ldots\} \text{ and } \mathrm{Elim}(t',\ldots)\{\cdots\} \text{ not in } \mathcal{O}^- \\ \mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \text{ and } \mathrm{Elim}(t' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\} \text{ are weak} \\ I, I' \text{ are weak inductive types} \end{array}}{\mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \approx_\Gamma \mathrm{Elim}(t' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\}}\ [\textsc{Elim}^{\mathcal{W}}]$$

$$\frac{\begin{array}{c} t = t' \quad I = I' \quad Q = Q' \quad \vec{v} = \vec{v}' \quad \vec{f} = \vec{f}' \\ \mathcal{T}, \mathrm{Eq}(\Gamma) \not\models 0 = 1 \text{ or } \mathrm{Elim}(t,\ldots)\{\ldots\} \text{ and } \mathrm{Elim}(t',\ldots)\{\cdots\} \text{ not in } \mathcal{O}^- \\ \mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \text{ is not weak} \\ \text{or } I \text{ or } I' \text{ is not a weak inductive type} \end{array}}{\mathrm{Elim}(t : I\,[\vec{v}] \to Q)\{\vec{f}\} \approx_\Gamma \mathrm{Elim}(t' : I'\,[\vec{v}'] \to Q')\{\vec{f}'\}}\ [\textsc{Elim}^{\mathcal{S}}]$$

$$\frac{A \approx_\Gamma A' \quad \vec{C} \approx_\Gamma \vec{C}'}{\mathrm{Ind}(X : A)\vec{C} \approx_\Gamma \mathrm{Ind}(X : A')\vec{C}'}\ [\textsc{Ind}] \qquad \frac{I \approx_\Gamma I' \quad \mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot}{I^{[n]} \approx_\Gamma I'^{[n]}}\ [\textsc{Constr}]$$

Figure 5.2: Conversion relation $\approx^?_\Gamma$ (Part. 2)

Correctness of extractable terms

Before proving correctness and completeness of $\approx_\Gamma$, we first must ensure that the sets $\mathcal{O}^+$, $\mathcal{O}^-$ and $\mathcal{P}^+$ conform to the required restrictions of our meta-theory.

The set $\mathcal{O}^+$ is clearly closed by reduction. It is also clear that $\mathcal{O}^+ \subseteq \mathcal{O}^-$ and that $\mathcal{O}^+$ and $\mathcal{O}^-$ are stable by well-sorted substitutions. Moreover, terms of $\mathcal{O}\backslash\mathcal{O}^-$ having empty algebraic caps, no R-reduction can occur at the root of any term of $\mathcal{O}^-$.

Likewise, **nat** being in $\mathcal{P}$ and conversion conserving classes, we have $\mathcal{P}^+ \subseteq \mathcal{P}$. Moreover, $\mathcal{P}^+$ is stable by conversion by definition.

Now, assume that $t \in \mathcal{O}^-$ and $u \in$ CCIC s.t. $t \leftrightarrow_* u$. Then, $t \to_* x\,t_1 \ldots t_n$ (resp. $t \to_* f\,t_1 \ldots t_n$ with $f\,t_1 \ldots t_n$ being not fully applied). Note that no reduction can occur at the root of $x\,t_1 \ldots t_n$ (resp. $f\,t_1 \ldots t_n$) since they are not algebraic. By confluence of $\to$, $u \to_* x\,t'_1 \ldots t'_n$ (resp. $u \to_* f\,t'_1 \ldots t'_n$) with for all i, $t_i \to_* t'_i$. Hence, $u \in \mathcal{O}^-$ and $\mathcal{O}^-$ is stable by $\to$-equivalence.

We are left to prove:

Lemma 5.6

Assume that $\mathcal{O}^- \times \mathcal{O}^- \nsubseteq \sim_\Gamma$. If $t \sim_\Gamma u$ with $t \in \mathcal{O}^+$, then $u \to_* u' \in \mathcal{O}^+$ and $t \sim_\Gamma u'$. (Resp., if $u \in \mathcal{O}^+$, then $t \to_* t' \in \mathcal{O}^+$ and $t' \sim_\Gamma u$)

Proof. By induction on the definition of $t \sim_\Gamma u$, we prove that:

1. If $t \in \mathcal{O}^+$, then $u \to_* u' \in \mathcal{O}^+$ and $t \sim_\Gamma u'$,
2. if $t = f\,t_1, \ldots, t_n$ with t not fully applied and $t_1, \ldots, t_n \in \mathcal{O}^+$, then $u \to_* f\,u_1, \ldots, u_n$, and for all i, $t_i \sim_\Gamma u_i$ and $u_i \to_* u'_i \in \mathcal{O}^+$.

From the form of t, only 5 rules are applicable: [Eq], [Rw-Right], [Refl], [Ded] and [$\textsc{App}^{\mathcal{W}}$]:

- [Eq]. We have $t, u \in \mathcal{O}^+$ by rule assumption.
- [Rw-Right]. Straightforward application of induction hypothesis.
- [Refl] - $t = u$. We take $u' = u$.
- [Ded]. $\overbrace{\{w_1 = w_2 \mid w_1 \sim_\Gamma w_2, w_1, w_2 \in \mathcal{O}^+\}}^{E} \vDash \mathcal{A}_{\sim_\Gamma}(t) = \mathcal{A}_{\sim_\Gamma}(u)$

  If $t \in \mathcal{O}^+$, then $\mathcal{A}_{\sim_\Gamma}(t) = t$. Since E is $\mathcal{T}$-valid (otherwise we would have $\mathcal{O}^- \times \mathcal{O}$-$\subseteq \sim_\Gamma$, which contradicts lemma assumption), u is algebraic from Lemma 5.5. Thus, $\mathcal{A}_{\sim_\Gamma}(u) = u$. Let $u' = u_{\downarrow_R} \in \mathcal{O}^+$. From Lemma 5.5, $\mathcal{T}, E \vDash t = u'$, and thus, $t \sim_\Gamma u'$.

  If $t = f\,t_1, \ldots, t_n$, then [Ded] is not applicable since t is not fully applied ($t \notin \mathcal{O}^-$).
- [App$^{\mathcal{W}}$]. $t = t_1\,t_2$, $u = u_1\,u_2$, $t_i \sim_\Gamma u_i$, t_1, t_2, u_1, u_2 are weak

  If $t \in \mathcal{O}^+$, then $t = f\,v_1\,\ldots\,v_n$ with all the v_i's in $\mathcal{O}^+$. Then $t_1 = f\,v_1, \ldots, v_{n-1}$ and $t_2 = v_n$ and from induction hypothesis, $u_1 \to_* p_1 = f\,w_1, \ldots, w_{n-1}$ and $u_2 \to_* p_2 = w_n$ with all the w_i's in $\mathcal{O}^+$ and for all i, $v_i \sim_\Gamma w_i$. Let $E = \{v_i = w_i\}_i$. Then, $\mathcal{T}, E \vDash t = p_1\,p_2$. From Lemma 5.5, $\mathcal{T}, E \vDash t = (p_1\,p_2)_{\downarrow_R}$. Hence, $t \sim_\Gamma (p_1\,p_2)_{\downarrow_R}$ by [Ded] with $u \to_* (p_1\,p_2)_{\downarrow_R} \in \mathcal{O}^+$.

  If $t_1\,t_2 = f\,p_1, \ldots, p_n$, then we conclude by a straightforward application of the induction hypothesis. □

Corollary 5.7

The reduction $\to$ is strongly normalizing on well-formed terms.

Correctness

We now state and prove the correctness of $\approx_\Gamma$:

Lemma 5.8 — **Correctness**

If $t \approx_\Gamma u$ then $t \sim_\Gamma u$.

Proof. The proof is done by induction on the definition of $t \approx_\Gamma u$:

- If $t \approx_\Gamma u$ by [Refl-$\star$], [Refl-$\square$] or [Refl-$\mathcal{X}$], we conclude that $t \sim_\Gamma u$ by [Refl].
- Assume that $t \approx_\Gamma u$ is obtained by rule [Unsat]. For any equation $w_1 = w_2 \in \mathrm{Eq}(\Gamma)$, we know that $w_1 \sim_\Gamma w_2$ by [Eq]. Thus, from $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \bot$, and $t, u \in \mathcal{O}^-$, we have $t \sim_\Gamma u$ by [Ded].
- If $t \approx_\Gamma u$ is obtained from [Lam], [Prod], [Elim$^{\mathcal{W}}$], [App$^{\mathcal{W}}$], [Eq], [Symb], [Ind] or [Constr], we conclude by direct application of the induction hypothesis, using the corresponding congruence rule of $\sim_\Gamma$.
- If $t \approx_\Gamma u$ is derived from [Elim$^{\mathcal{S}}$] or [App$^{\mathcal{S}}$], then $t = u$. Hence, $t \sim_\Gamma u$ by [Refl].
- If $t \approx_\Gamma u$ is obtained from [Ded], with $t = C_t[a_1, \ldots, a_k]$ and $u = C_u[a_{k+1}, \ldots, a_{k+l}]$, C_t, C_u being maximal algebraic contexts, then let $c_1, \ldots, c_{k+l} \in \mathcal{Y}$ be the variables affected to $a_1, \ldots, a_{k+l}$ in the application of the [Ded] rule. By application of the induction hypothesis, we know that if $a_i \approx_\Gamma a_j$, then $a_i \sim_\Gamma a_j$. Thus, the algebraic context being maximal, there exists a variables ξ injection (whose domain is

$\{c_1, \dots, c_{k+l}\})$ s.t. $\mathcal{A}_{\approx_\Gamma}(t) = C_t[c_1, \dots, c_k]\xi$ and $\mathcal{A}_{\approx_\Gamma}(u) = C_u[c_{k+1}, \dots, c_{k+l}]\xi$. Hence, $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \mathcal{A}_{\approx_\Gamma}(t) = \mathcal{A}_{\approx_\Gamma}(u)$ - variables of $\mathrm{Eq}(\Gamma)$ being disjoint from $\mathrm{dom}(\xi)$.

Now, by application of the [Eq] rule, we have $w_1 \sim_\Gamma w_2$ for any equation $w_1 = w_2 \in \mathrm{Eq}(\Gamma)$. Hence, by [Ded], $t \sim_\Gamma u$. □

Completeness

For completeness, we need to prove for $\approx_\Gamma$ lemmas similar to the Lemmas 4.20 and 4.22 for $\equiv_\Gamma$.

We start with some technical lemmas about $\approx_\Gamma$.

Lemma 5.9 — **Reflexivity**

For any term t and typing environment Γ, $t \approx_\Gamma t$.

Proof. Straightforward induction on the structure of t. □

Lemma 5.10 — **Preservation of classes**

1. The relation $\approx_\Gamma$ preserve classes.
2. If $t \in \mathcal{O}^-$ and $t \approx_\Gamma u$, then $u \in \mathcal{O}^-$.

Proof.

1. Straightforward by induction on the definition of $\approx_\Gamma$.
2. We first show that $\mathcal{O}\backslash\mathcal{O}^-$ is stable by $\approx_\Gamma$: by induction on the definition of $t \approx_\Gamma u$, we prove that if $t = x\, t_1, \dots t_n$ with $x \in \mathcal{X}\backslash\mathcal{X}^-_\star$ (resp. $t = f\, t_1, \dots t_n$ and not fully applied), then $u = x\, t_1, \dots t_n$ (resp. $u = f\, u_1, \dots u_n$)

   Only three rules are applicable: [Refl-$\mathcal{X}$], [Refl-Σ], [App]. The cases [Refl-$\mathcal{X}$] and [Refl-Σ] are immediate. We detail the [App] one. We have then $t = f\, t_1 \dots t_{n-1}\, t_n$ and $u = w\, w'$ with $f\, t_1 \dots t_{n-1} \approx_\Gamma w$ and $t_n \approx_\Gamma w'$. By application of the induction hypothesis, $w = f\, w_1 \dots w_{n-1}$ for some w_i's. Hence, $u = f\, w_1 \dots w_{n-1}\, w'$.

   The case $t = x\, t_1, \dots t_n$ is identical.

   Now, since $\mathcal{O}$ and $\mathcal{O}\backslash\mathcal{O}^-$ are stable by $\approx_\Gamma$, so is $\mathcal{O}^-$. □

Lemma 5.11 — **Stability of $\approx_\Gamma$**

If Γ and Δ are two typing environments s.t. $\mathrm{Eq}(\Gamma) = \mathrm{Eq}(\Delta)$, then $\approx_\Gamma = \approx_\Delta$.

Proof. Straightforward induction on the definition of $\approx_\Gamma$ (resp. $\approx_\Delta$). □

Lemma 5.12

If $t \approx_\Gamma u$ for $t, u \in \mathcal{O}^+$, then $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash t = u$.

Proof. If $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \bot$, then necessarily $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash t = u$. Otherwise, the only applicable rule is [Ded]. Since t and u are pure algebraic, we have $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash t = u$ from [Ded] assumptions. □

We now state and prove all the lemmas about stability of weak conversion we will need in completeness proof.

Lemma 5.13

Let $t, u \in \mathcal{WT}$ s.t. $t \approx_\Gamma u$ and two substitutions $\theta, \theta' : \Gamma \hookrightarrow \Delta$ s.t.

- $\mathrm{dom}(\theta) = \mathrm{dom}(\theta')$,
- for all $x \in \mathrm{dom}(\theta)$, $x\theta \approx_\Delta x\theta'$.

Then, $t\theta \approx_\Delta u\theta'$.

Proof. By induction on the definition $t \approx_\Gamma u$, we prove that for any substitutions θ, θ' conforming to the lemma assumptions, $t\theta \approx_\Delta u\theta'$:

- [Ded]. $t = C_t[a_1, \ldots, a_n]$, $u = C_u[a_{n+1}, \ldots, a_{n+k}]$,
  $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_t[y_1, \ldots, y_n] = C_u[y_{n+1}, \ldots, y_{n+k}]$, $a_i \approx_\Gamma a_j \Rightarrow y_i = y_j$

  Let $C_{t\theta}$ and $C_{u\theta}$ be the maximal algebraic caps of $t\theta$ and $u\theta$: $t\theta = C_{t\theta}[b_1, \ldots, b_l]$ and $u\theta = C_{u\theta}[b_{l+1}, \ldots, b_{l+p}]$. Let $z_1, \ldots, z_{l+p}$ be variables of $\mathcal{Y}$ s.t. if $b_i \approx_\Delta b_j$ then $z_i = z_j$.

  Since $C_{t\theta}$ and $C_{u\theta}$ span the contexts C_t and C_u, for all i, there exists a maximal algebraic context C_i s.t.

  $$a_i\theta = C_i[b_{\xi_i(1)}, \ldots, b_{\xi_i(\alpha_i)}].$$

  Assume a $\mathcal{T}$-model $\mathcal{M}$ and a $\mathcal{M}$-interpretation $\mathcal{I}$ s.t. $[\![\mathrm{Eq}(\Delta)]\!]^{\mathcal{I}}_{\mathcal{M}} = \top$ and

  $$[\![C_{t\theta}[z_1, \ldots, z_l]]\!]^{\mathcal{I}}_{\mathcal{M}} \neq [\![C_{u\theta}[z_{l+1}, \ldots, z_{l+p}]]\!]^{\mathcal{I}}_{\mathcal{M}}.$$

  We define the $\mathcal{M}$-interpretation $\mathcal{I}'$ as:

  $$\begin{array}{lll} \mathcal{I}'(y_i) = [\![C_i[z_{\xi_i(1)}, \ldots, z_{\xi_i(\alpha_i)}]]\!]^{\mathcal{I}}_{\mathcal{M}} & & \\ \mathcal{I}'(y) = \mathcal{I}(y) & & \text{if } y \in \mathcal{Y}\backslash\{y_1, \ldots, y_{n+k}\} \\ \mathcal{I}'(x) = [\![x\theta]\!]^{\mathcal{I}}_{\mathcal{M}} & & \text{if } x \in \mathcal{X}^-_\star \end{array}$$

  We first prove that if $y_i = y_j$, then

  $$[\![C_i[z_{\xi_i(1)}, \ldots, z_{\xi_i(\alpha_i)}]]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![C_j[z_{\xi_j(1)}, \ldots, z_{\xi_j(\alpha_j)}]]\!]^{\mathcal{I}}_{\mathcal{M}}$$

  hence assuring the well-formation of the definition of $\mathcal{I}'$.

  Assume that $a_i \approx_\Gamma a_j$. Without loss of generality, we here assume $i \leqslant n$ and $j > n$.

  From the induction hypothesis, $a_i\theta \approx_\Delta a_j\theta'$. Then,

  - If C_i and C_j are empty algebraic contexts, $a_i\theta = b_{\xi_i(1)}$ and $a_j\theta' = b_{\xi_j(1)}$. Thus, $z_{\xi_i(1)} = z_{\xi_j(1)}$ and we obtain the desired result.
  - Otherwise, by inversion of $a_i\theta \approx_\Delta a_j\theta'$, we have:

    $$\mathcal{T}, \mathrm{Eq}(\Delta) \vDash C_i[x_{\xi_i(1)}, \ldots, x_{\xi_i(\alpha_i)}] = C_j[x_{\xi_j(1)}, \ldots, x_{\xi_j(\alpha_j)}]$$

    for some fresh variables $\vec{x} \in \mathcal{Y}$ s.t. $b_i \approx_\Gamma b_j$ implies $x_i = x_j$. Since no variables of $\mathcal{Y}$ appear in $\mathrm{Eq}(\Delta)$, we obtain the desired result by a renaming from $\vec{x}$ to $\vec{z}$.

  Now,

  1. If $t \in \mathcal{O}^+$, then $[\![t]\!]^{\mathcal{I}'}_{\mathcal{M}} = [\![t\theta]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![t\theta']\!]^{\mathcal{I}}_{\mathcal{M}}$. Indeed, if $t = x \in \mathcal{X}^-_\star$, $[\![t]\!]^{\mathcal{I}'}_{\mathcal{M}} = \mathcal{I}'(x) = [\![x\theta]\!]^{\mathcal{I}}_{\mathcal{M}}$ by definition of $\mathcal{I}'$. Furthermore, since $x\theta \approx_\Delta x\theta'$ and $x\theta$ and $x\theta' \in \mathcal{O}^+$, by Lemma 5.12, $\mathcal{T}, \mathrm{Eq}(\Delta) \vDash_{\mathcal{M},\mathcal{I}} x\theta = x\theta'$. Hence, having $[\![\mathrm{Eq}(\Delta)]\!]^{\mathcal{I}}_{\mathcal{M}} = \top$, we obtain $[\![x\theta]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![x\theta']\!]^{\mathcal{I}}_{\mathcal{M}}$.

2. We also have $[\![C_{t\theta}[z_1,\dots,z_l]]\!]^{\mathcal{J}'}_{\mathcal{M}} = [\![C_t[y_1,\dots,y_n]]\!]^{\mathcal{J}}_{\mathcal{M}}$
$[\![C_{u\theta}[z_{l+1},\dots,z_{l+p}]]\!]^{\mathcal{J}'}_{\mathcal{M}} = [\![C_u[y_{n+1},\dots,y_{n+k}]]\!]^{\mathcal{J}}_{\mathcal{M}}$

   This is proved by a straightforward induction on the algebraic structure of C_t and $C_{t\theta}$ (resp. C_u and $C_{u\theta}$). Notably, if $C_t[y_1,\dots,y_n] = y_i$ for some i, then $C_{t\theta}[z_1,\dots,z_l]$ is necessarily of the form $C_i[z_{\xi_i(1)},\dots,z_{\xi_i(\alpha_i)}]$ by definition of $C_{t\theta}$ and we obtain the desired result from the definition of $\mathcal{J}'$.

Thus, $[\![\mathrm{Eq}(\Gamma)]\!]^{\mathcal{J}'}_{\mathcal{M}} = \top$ and $[\![C_t[y_1,\dots,y_n]]\!]^{\mathcal{J}'}_{\mathcal{M}} \neq [\![C_u[y_{n+1},\dots,y_{n+k}]]\!]^{\mathcal{J}'}_{\mathcal{M}}$. This contradicts $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_t[y_1,\dots,y_n] = C_u[y_{n+1},\dots,y_{n+k}]$.

Hence $\mathcal{T}, \mathrm{Eq}(\Delta) \vDash C_{t\theta}[z_1,\dots,z_l] = C_{u\theta}[z_{l+1},\dots,z_{l+p}]$ and by [Ded], $t\theta \approx_\Delta u\theta'$.

- [Unsat]. As for the previous case, if we have a $\mathcal{T}$-model $\mathcal{M}$ and a $\mathcal{J}$ interpretation s.t. $[\![\mathrm{Eq}(\Delta)]\!]^{\mathcal{J}}_{\mathcal{M}} = \top$, we construct a $\mathcal{M}$-interpretation $\mathcal{J}'$ as

$$\begin{aligned} \mathcal{J}'(y) &= \mathcal{J}(y) && \text{if } y \in \mathcal{Y} \\ \mathcal{J}'(x) &= [\![x\theta]\!]^{\mathcal{J}}_{\mathcal{M}} && \text{if } x \in \mathcal{X}^-_\star \end{aligned}$$

  which is s.t. $[\![\mathrm{Eq}(\Gamma)]\!]^{\mathcal{J}'}_{\mathcal{M}} = \top$, which contradicts $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \bot$.

  Hence, $\mathcal{T}, \mathrm{Eq}(\Delta) \vDash \bot$ and, $t\theta, u\theta'$ begin in $\mathcal{O}^-$, $t\theta \approx_\Delta u\theta'$ by [Unsat]

- [Refl-s]. Immediate since then $t\theta = t$ and $u\theta = u$.

- [Refl-$\mathcal{X}$] - $t = u = x \in \mathcal{X}$. If $x \in \mathrm{dom}(\theta)$, then $x\theta \approx_\Delta x'\theta$ by assumption. Otherwise, $x\theta = x \approx_\Delta x = x\theta'$

- [App$^{\mathcal{W}}$]. $t = t_1\, t_2$, $u = u_1\, u_2$, $t_i \approx_\Gamma u_i$

  If $\mathcal{T}, \mathrm{Eq}(\Delta) \vDash \bot$ and $t\theta, u\theta \in \mathcal{O}^-$, we conclude by application of the [Unsat] rule. Otherwise, $t_i\theta, u_i\theta$ are in $\mathcal{WT}$ since co-domains of θ, θ' are uniquely composed of weak terms, and we conclude by direct application of induction hypothesis.

- [App$^{\mathcal{S}}$]. Not applicable since $t, u \in \mathcal{WT}$

- [Elim$^{\mathcal{W}}$]. $t = \mathrm{Elim}(v : I\,[\vec{w}] \to Q)\{\vec{f}\}$, $u = \mathrm{Elim}(v' : I'\,[\vec{w}'] \to Q)\{\vec{f}\,'\}$, $v = v'$, $Q \approx_\Gamma Q'$, $I \approx_\Gamma I'$, $\vec{w} \approx_\Gamma \vec{w}'$ and $\vec{f} \approx_\Gamma \vec{f}\,'$

  If $\mathcal{T}, \mathrm{Eq}(\Delta) \vDash \bot$ and $t, u \in \mathcal{O}^-$, then we conclude by application of the [Unsat] rule. Otherwise, since $t, u \in \mathcal{WT}$, $v\theta = v = v' = v'\theta'$. We then conclude by application of induction hypothesis and [Elim$^{\mathcal{W}}$].

- [Elim$^{\mathcal{S}}$]. Not application since $t, u \in \mathcal{WT}$.

- If $t \approx_\Gamma u$ by [Eq], [Symb], [Constr] or [Ind], we conclude by a application of the induction hypothesis. □

Lemma 5.14

Suppose that $t \approx_\Gamma u$ and t and u do not contain subterms of the form $a\,v$ with a algebraic. If $t \to t'$, then there exists a term u' s.t. $u \to_\leqslant u'$ and $t' \approx_\Gamma u'$. Symmetrically, if $u \to u'$, then there exists a term t' s.t. $t \to_\leqslant t'$ and $t' \approx_\Gamma u'$.

Proof. By induction on the definition of $t \approx_\Gamma u$.

- [Unsat]. We simply take $u' = u$.

- $[\textsc{App}^{\mathcal{W}}]$. $t = t_1\,t_2$, $u = u_1\,u_2$, $t_i \approx_\Gamma u_i$, $t_1, t_2, u_1, u_2 \in \mathcal{WT}$

  If reduction occurs in t_1 or t_2, we conclude by application of induction hypothesis.

  If reduction occurs at root position, this cannot be a ι-reduction since t is headed with an application, nor a $\mathcal{R}$-reduction since t cannot have an algebraic cap from rule assumptions. Thus, $t_1 = \lambda[x :^u T].\, v$ and $t \to v\{x \mapsto t_2\}$.

  Now, since t_1 is headed with a lambda abstraction, t_1 can be converted to u_1 by only three rules: $[\textsc{Unsat}]$, $[\textsc{Ded}]$ and $[\textsc{Lam}]$. If $[\textsc{Unsat}]$ applies, then $t, u \in \mathcal{O}^-$ and $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \bot$, which contradicts lemma assumptions. If $[\textsc{Ded}]$ applies, then u contains a subterm of the form $a\,v$ with a algebraic, which contradicts rule assumption. Then, $u_1 = \lambda[x :^u U].\, w$ with $T \approx_\Gamma U$ and $v \approx_{\Gamma,[x:^u T]} w$ and $u \to w\{x \mapsto u_2\}$. Since T is marked with the unrestricted annotation in $\Gamma, [x :^u T]$, we have $\mathrm{Eq}(\Gamma) = \mathrm{Eq}(\Gamma, [x :^u T])$ and thus $v \approx_\Gamma w$. We conclude $v\{x \mapsto t_2\} \approx_\Gamma w\{x \mapsto u_2\}$ by Lemma 5.13.

- $[\textsc{Elim}^{\mathcal{W}}]$. $t = \mathrm{Elim}(v : I\,[\vec{w}] \to Q)\{\vec{f}\}$, $u = \mathrm{Elim}(v' : I'\,[\vec{w}'] \to Q')\{\vec{f}'\}$, $v = v'$

  The proof is identical to the $[\textsc{Elim}^{\mathcal{W}}]$ case of Lemma 4.22.

- If $t \approx_\Gamma u$ is obtained from $[\textsc{App}^{\mathcal{S}}]$ or $[\textsc{Elim}^{\mathcal{S}}]$, then we take $u' = u$.

- $[\textsc{Prod}]$. $t = \forall(x :^a T).\, v$, $u = \forall(x :^a U).\, w$, $T \approx_\Gamma U$, $v \approx_{\Gamma,[x:^a T]} w$

  If reduction occurs on v, we conclude by application of the induction hypothesis. If $T \to T'$, then by application of induction hypothesis, there exists a term U' s.t. $U \to_{\leqslant} U'$ and $T' \approx_\Gamma U'$. Since, $\mathrm{Eq}(\Gamma, [x :^a T]) = \mathrm{Eq}(\Gamma, [x :^a T'])$, then $v \sim_{\Gamma,[x:^a T']} w$. Thus, by application of $[\textsc{Prod}]$, $\forall(x :^a T').\, v \approx_\Gamma \forall(x :^a U').\, w$.

- $[\textsc{Lam}]$. As for the $[\textsc{Prod}]$ case.

- All other cases are done by direct application of induction hypothesis. □

We can now state and prove completeness of weak conversion on strongly normalizing terms. Note that we also require terms to have no over-applied algebraic subterms, which is always the case for well-formed terms.

Lemma 5.15 — **Completeness**

Let $t, u \in \mathcal{SN}$ and Γ a typing environment. Suppose that t and u cannot reduce to a term containing a subterm of the form $a\,v$ with a having a non-empty algebraic cap.

If $t \sim_\Gamma u$, then $t \xrightarrow{!} \approx_\Gamma \xleftarrow{!} u$.

Proof. By induction on the definition of $t \sim_\Gamma u$:

- $[\textsc{Eq}]$ - $[x :^r T] \in \Gamma$, $T \to_* t \doteq u$ with $t, u \in \mathcal{O}^+$. If $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \bot$, then $t \approx_\Gamma u$ by $[\textsc{Unsat}]$. Otherwise, $t = u \in \mathrm{Eq}(\Gamma)$ and $t \approx_\Gamma u$ by $[\textsc{Ded}]$.

- $[\textsc{Ded}]$. $\underbrace{\{w_1 = w_2 \mid w_1, w_2 \in \mathcal{O}^+, w_1 \sim_\Gamma w_2\}}_{E} \vDash \mathcal{A}_{\sim_\Gamma}(t) = \mathcal{A}_{\sim_\Gamma}(u)$, $t, u \in \mathcal{O}^-$

  Let $w_1 = w_2 \in E$. By application of the induction hypothesis and Lemma 5.12, $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash w_1 = w_2$. Thus, $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash E$. Now,

  - If $\mathcal{T}, E \vDash \bot$, then $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \bot$ and we conclude by $[\textsc{Unsat}]$.
  - Otherwise, if t nor u has a non-empty algebraic cap, then $\mathcal{A}_{\sim_\Gamma}(t) = y_1 \in \mathcal{Y}$, $\mathcal{A}_{\sim_\Gamma}(u) = y_2 \in \mathcal{Y}$ and $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash y_1 = y_2$. Since $\mathcal{T}, \mathrm{Eq}(\Gamma) \nvDash \bot$ and no variables of $\mathcal{Y}$ occurs in $\mathrm{Eq}(\Gamma)$, then $y_1 = y_2$. Thus, we can use the induction hypothesis on $t \sim_\Gamma u$ and obtained the desired result.

- Otherwise, let $t = C_t[a_1, \dots, a_n]$ and $u = C_u[a_{n+1}, \dots, a_{n+k}]$ where C_t and C_u are maximal algebraic caps. let $\vec{y} \in \mathcal{Y}$ s.t. if $a_{i\downarrow} \approx_\Gamma a_{j\downarrow}$, then $y_i = y_j$. Since $a_i \sim_\Gamma a_j$ implies $a_{i\downarrow} \approx_\Gamma a_{j\downarrow}$ and $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \mathcal{A}_{\sim_\Gamma}(t) = \mathcal{A}_{\sim_\Gamma}(u)$, we have $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_t[y_1, \dots, y_n] = C_u[y_{n+1}, \dots, y_{n+k}]$. Now, from Lemma 5.5, we have:
$$\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_{t\downarrow_R}[y_1, \dots, y_n] = C_{u\downarrow_R}[y_{n+1}, \dots, y_{n+k}]$$
Hence, $t_\downarrow \approx_\Gamma u_\downarrow$.

- [App$^{\mathcal{W}}$]. $t = t_1\, t_2$, $u = u_1\, u_2$, $t_i \sim_\Gamma u_i$, $t_1, t_2, u_1, u_2 \in \mathcal{WT}$

  If $t, u \in \mathcal{O}^-$ and $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \bot$, then $t \approx_\Gamma u$ by [Unsat].

  Otherwise, by application of the induction hypothesis, $t_{1\downarrow} \approx_\Gamma u_{1\downarrow}$ and $t_{2\downarrow} \approx_\Gamma u_{2\downarrow}$. Applying [App], $t_{1\downarrow}\, t_{2\downarrow} \approx_\Gamma u_{1\downarrow}\, u_{2\downarrow}$. By multiple applications of Lemma 5.14 along the reduction path $t_{1\downarrow}\, t_{2\downarrow} \xrightarrow{!} (t_1\, t_2)_\downarrow$, there exists a term v s.t. $(t_1\, t_2)_\downarrow \approx_\Gamma v$ and $u \to_* v$. Then, by multiple applications of Lemma 5.14 along the path $v \xrightarrow{!} v_\downarrow$, we obtain $(t_1\, t_2)_\downarrow \approx_\Gamma (u_1\, u_2)_\downarrow$.

- [Refl] - $t = u$. We conclude by Lemma 5.9.

- [Prod]. $t = \forall(x :^a T).\, v$, $u = \forall(x :^a U).\, w$, $T \sim_\Gamma U$, $v \sim_{\Gamma,[x:^a T]} w$

  By induction hypothesis, $T \approx_\Gamma U$ and $t \approx_{\Gamma,[x:^a T]} u$. Then $\forall(x :^a T).\, v \approx_\Gamma \forall(x :^a U).\, w$ by [Prod].

- [Lam]. $t = \lambda[x :^a T].\, v$, $u = \lambda[x :^a U].\, w$, $T \sim_\Gamma U$, $v \sim_{\Gamma,[x:^a T]} w$

  If $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \bot$ and $t, u \in \mathcal{O}^-$, we conclude by [Unsat]. Otherwise, we conclude by application of the induction hypothesis as in the [Prod] case.

- [Elim$^{\mathcal{W}}$]. $t = \mathrm{Elim}(v : I\,[\vec{w}] \to Q)\{\vec{f}\}$, $u = \mathrm{Elim}(v' : I'\,[\vec{w}'] \to Q')\{\vec{f}'\}$,
  $v \leftrightarrow_* v'$, $t, u \in \mathcal{WT}$

  By application of the induction hypothesis, we have $I_\downarrow \approx_\Gamma I'_\downarrow, \cdots, \vec{f}_\downarrow \approx_\Gamma \vec{f}'_\downarrow$. Thus, $t' \approx_\Gamma u'$ where
  $$t' = \mathrm{Elim}(v_\downarrow : I_\downarrow\,[\vec{w}_\downarrow] \to Q_\downarrow)\{\vec{f}_\downarrow\}$$
  $$u' = \mathrm{Elim}(v'_\downarrow : I'_\downarrow\,[\vec{w}'_\downarrow] \to Q'_\downarrow)\{\vec{f}'_\downarrow\}$$
  As for the [App$^{\mathcal{W}}$] case, we conclude by application of Lemma 5.14 on the reduction paths $t' \to_* t'_\downarrow$ and $u' \to_* u'_\downarrow$. □

Decidability of conversion

Before terminating the proof, we are left to prove that $\approx_\Gamma$ is an equivalence relation on well-formed terms.

Lemma 5.16

If $T \to_* t \doteq u$ with $t, u \in \mathcal{O}^+$ and $T \approx_\Gamma U$, then $U \to_* t' \doteq u'$ with $t', u' \in \mathcal{O}^+$, $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash t_{\downarrow_R} = t'_{\downarrow_R}$ and $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash u_{\downarrow_R} = u'_{\downarrow_R}$.

Proof. By correctness, $t \sim_\Gamma u$. By product compatibility, $U \to_* t_1 \doteq u_1$ with $t \sim_\Gamma t_1$ and $u \sim_\Gamma u_1$. By assumption on $\mathcal{O}+$, there exist $t', u' \in \mathcal{O}^+$ s.t. $t_1 \to_* t'$, $u_1 \to_* u'$, $t \sim_\Gamma t'$ and $u \sim_\Gamma u'$. By Lemma 5.15, $t \approx_\Gamma t'$ and $u \approx_\Gamma u'$. By Lemma 5.12, $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash t_{\downarrow_R} = t'_{\downarrow_R}$ and $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash u_{\downarrow_R} = u'_{\downarrow_R}$. □

Corollary 5.17

If $\Gamma = \Gamma_1, [x :^a T], \Gamma_2$ and $\Delta = \Gamma_1, [x :^a T'], \Gamma_2$ with $T \approx_{\Gamma_1} T'$, then $\approx_\Gamma = \approx_\Delta$.

Proof. Straightforward induction, using $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash \mathrm{Eq}(\Delta)$ and $\mathcal{T}, \mathrm{Eq}(\Delta) \vDash \mathrm{Eq}(\Gamma)$. □

We now show that $\approx_\Gamma$ is an equivalence:

Lemma 5.18

i) If $t \approx_\Gamma u \approx_\Gamma v$, then $u \approx_\Gamma v$. ii) If $t \approx_\Gamma u$, then $u \approx_\Gamma t$.

Proof. From now on, we write $w_1 \approx_\Gamma^S w_2$ for $w_1 \approx_\Gamma w_1$ or $w_2 \approx_\Gamma w_2$.

By induction on the definition of $t \approx_\Gamma u$, we prove: i) $u \approx_\Gamma t$, ii) if $u \approx_\Gamma^S v$, then $t \approx_\Gamma v$, iii) if $v \approx_\Gamma^S t$, then $v \approx_\Gamma t$.

We only details interesting cases, i.e. i) $u \approx_\Gamma t$ for the rules [Prod] and [Lam], ii) if $u \approx_\Gamma v$, then $t \approx_\Gamma v$ for all the rules.

- [Unsat]. If $u \approx_\Gamma v$, then $v \in \mathcal{O}^-$ by Lemma 5.10. Then, $t \approx_\Gamma v$ by [Unsat].
- [Ded]. $t = C_1[t_1, \dots, t_n]$, $u = C_2[u_1, \dots, u_k]$

  We have $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_1[x_1, \dots, x_n] = C_2[y_1, \dots, y_k]$ with i) $x_i = x_j$ if $t_i \approx_\Gamma t_j$, ii) $y_i = y_j$ if $u_i \approx_\Gamma u_j$, and iii) $x_i = y_j$ if $t_i \approx_\Gamma^S u_j$.

  - If u or v has a non-empty algebraic cap, then $u \approx_\Gamma v$ by [Ded], i.e.

    $$v = C_3[v_1, \dots, v_p]$$
    $$\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_2[y'_1, \dots, y'_k] = C_3[z_1, \dots, z_p]$$

    with $y'_1, \dots, y'_p, z_1, \dots, z_k$ fresh variables of $\mathcal{Y}$ and i) $y'_i = y'_j$ if $u_i \approx_\Gamma u_j$, ii) $z_i = z_j$ if $v_i \approx_\Gamma v_j$, and iii) $y'_i = z_j$ if $u_i \approx_\Gamma^S v_j$.

    Let $x'_1, \dots, x'_n, z'_1, \dots z'_p, y''_1, \dots, y''_k$ s.t.

    $$x'_i = x'_j \quad \text{if } t_i \approx_\Gamma t_j \qquad z'_i = z'_j \quad \text{if } v_i \approx_\Gamma v_j$$
    $$x'_i = y''_j \quad \text{if } t_i \approx_\Gamma^S u_j \qquad y''_i = z'_j \quad \text{if } u_i \approx_\Gamma^S v_j$$

    Let $\theta : \mathcal{Y} \to \mathcal{Y}$ defined by

    $$\theta = \{\overrightarrow{x_i \mapsto x'_i}\} \cup \{\overrightarrow{y_i \mapsto y''_i}\} \cup \{\overrightarrow{y'_i \mapsto y''_i}\} \cup \{\overrightarrow{z_i \mapsto z'_i}\}$$

    One can check that the definition is well-formed since the conditions equating the variables of the co-domain are stronger that the one equating the variables of the domain. Hence,

    $$\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_1[x_1, \dots, x_n]\theta = C_2[y_1, \dots, y_k]\theta$$
    $$\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_2[y'_1, \dots, y'_k]\theta = C_2[z_1, \dots, z_p]\theta$$

    Since $C_2[y_1, \dots, y_k]\theta = C_2[y'_1, \dots, y'_k]\theta$,

    $$\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C_1[x'_1, \dots, x'_n] = C_3[z'_1, \dots, z'_p].$$

    Now, if $t_i \approx_\Gamma^S u_{j_1} \approx_\Gamma^S \cdots u_{j_\iota} \approx_\Gamma^S v_q$, then by repeated application of the induction hypothesis on $u_{j_{\iota-1}} \approx_\Gamma u_{j_\iota}, \dots, u_{j_1} \approx_\Gamma^S u_{j_2}$, we have $u_{j_\iota} \approx_\Gamma^S v_q, \dots, u_{j_1} \approx_\Gamma^S v_q$. By a last application of the induction hypothesis, we obtain $t_i \approx_\Gamma^S v_q$.

    Thus, we could have defined $x'_1, \dots, x'_n, z'_1, \dots z'_p$ as being fresh variables of $\mathcal{Y}$ s.t.

$$\text{i) } x'_i = x'_j \text{ if } t_i \approx_\Gamma t_j, \text{ ii) } z'_i = z'_j \text{ if } v_i \approx_\Gamma v_j, \text{ and iii) } x'_i = z'_j \text{ if } u_i \approx^S_\Gamma v_j.$$

Hence, $t \approx_\Gamma v$ by [Ded].

- Otherwise, u and v have empty-algebraic caps. Hence, $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C[x_1, \dots, x_n] = y$ with $x_1, \dots, x_n, y$ s.t. i) $x_i = x_j$ if $t_i \approx_\Gamma t_j$, and ii) $x_i = y$ if $t_i \approx^S_\Gamma u$.
  Let $x'_1, \dots, x'_n, y'$ s.t.: i) $x'_i = x'_j$ if $t_i \approx_\Gamma t_j$, and ii) $x'_i = y'$ if $t_i \approx^S_\Gamma v$.
  By application of the induction hypothesis, we have $t_i \approx^S_\Gamma u$ implies $t_i \approx^S_\Gamma v$. Thus, the mapping $\theta = \{x_i \mapsto x'_i\}_i \cup \{y \mapsto y'\}$ is a valid substitution and:
  $$\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash C[x_1, \dots, x_n]\theta = y\theta$$
  Hence, $t \approx_\Gamma v$ by rule [Ded].

- [App$^{\mathcal{W}}$]. $t = t_1\, t_2$, $u = u_1\, u_2$, $t_i \approx_\Gamma u_i$ and t_1, t_2, u_1, u_2 are weak
  $u \approx_\Gamma v$ can be derived by [Ded], or [App$^{\mathcal{W}}$], or [App^{S}] or [Unsat].

  - [Ded]. We have $v = C[v_1, \dots, v_k]$ and $\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash y = C[z_1, \dots, z_k]$ where $y, z_1, \dots, z_k$ are s.t. i) $z_i = z_j$ if $v_i \approx_\Gamma v_j$, and ii) $y = z_i$ if $u \approx^S_\Gamma v_i$.
    Let $y', z'_1, \dots, z'_k$ s.t. i) $z'_i = z'_j$ if $v_i \approx_\Gamma v_j$, and ii) $y' = z'_i$ if $t \approx^S_\Gamma v_i$.
    By application of the induction hypothesis, we have $u \approx^S_\Gamma v_i$ implies $t \approx^S_\Gamma v_i$. Thus, the mapping $\theta = \{\overrightarrow{z_i \mapsto z'_i}\} \cup \{y \mapsto y'\}$ is a valid substitution and:
    $$\mathcal{T}, \mathrm{Eq}(\Gamma) \vDash y\theta = C[z_1, \dots, z_n]\theta$$
    Hence, $t \approx_\Gamma v$ by rule [Ded].
  - [App$^{\mathcal{W}}$]. Straightforward application of the induction hypothesis.
  - [App^{S}]. Straightforward since $u = v$.
  - [Unsat]. Then, $u \in \mathcal{O}^-$ and so is t by Lemma 5.10, which contradicts [App$^{\mathcal{W}}$] assumptions.

- The cases for rules [Eq], [Symb], [Ind] [Constr], [Elim$^{\mathcal{W}}$], [Refl-?] are similar to the [App$^{\mathcal{W}}$] one.
- The cases [App^{S}] and [Elim^{S}] are straightforward since $t = u$.
- [Lam]. The proof of transitivity is as for the rule [App$^{\mathcal{W}}$]. We here detail the proof of symmetry. We have $t = \lambda[x :^a T].v$ and $u = \lambda[x :^a U].w$ with $T \approx_\Gamma U$ and $v \approx_{\Gamma,[x:^a T]} w$. By application of the induction hypothesis, $U \approx_\Gamma T$ and $w \approx_{\Gamma,[x:^a T]} v$. By Lemma 5.17, $w \approx_{\Gamma,[x:^a U]} v$. Hence, $u \approx_\Gamma t$ by [Lam].
- [Prod]. As for the [Lam]. □

Corollary 5.19

The relation $\sim_\Gamma$ is symmetric and transitive on well-formed terms.

Proof. 1. Suppose that $t \sim_\Gamma u \sim_\Gamma v$, with t, u, v well-formed. Since a well formed term (and any of its reduces by subject reduction) cannot contain a subterm of the form $a\,v$ with a algebraic, by Lemma 5.15, $t_\downarrow \approx_\Gamma u_\downarrow \approx_\Gamma v_\downarrow$. By Lemma 5.18, $t_\downarrow \approx_\Gamma v_\downarrow$, and by Lemma 5.8, $t_\downarrow \sim_\Gamma v_\downarrow$. Hence, by multiple application of the rule [Rw], $t \sim_\Gamma v$.

2. Similar to the proof of transitivity. □

Theorem 5.1

Suppose that $\approx_\Gamma$ is decidable. The relation $\sim_\Gamma$ for Γ well-formed is decidable on well-formed terms.

Proof. Let t and u be two well-formed terms under Γ.

Suppose first that $t \sim_\Gamma u$. Since a well formed term (and any of its reduces by subject reduction) cannot contain a subterm of the form $a\,v$ with a algebraic (since algebraic terms have non-functional types) and since $\to$ is strongly normalizing on well-formed terms, we conclude by Lemma 5.15 that $t_\downarrow \approx_\Gamma u_\downarrow$.

Conversely, if $t_\downarrow \approx_\Gamma u_\downarrow$, then $t_\downarrow \sim_\Gamma u_\downarrow$ by Lemma 5.8. Using multiple applications of [Rw] yield $t \sim_\Gamma u$. □

Corollary 5.20

Assume that Γ is a well-formed environment. Then, the relation $\sim_\Gamma$ is decidable on well--formed terms.

Proof. We prove that $\approx_\Gamma$ is decidable. A straightforward induction on $t \approx_\Gamma u$ for t, u well--formed shows that we only need to compute $\mathrm{Eq}(\Delta)$ for well-formed typing environments Δ. From the definition of $\mathrm{Eq}(\Delta)$ and strong normalization of well-formed terms, we have:

$$\mathrm{Eq}(\Delta) = \{t = u \mid [x :^u T] \in \Gamma, T_\downarrow = (t \doteq u)\}$$

Rules of $\approx_\Gamma$ being structural, there is a simple top-down algorithm checking conversion: assume that t and u are two terms well formed under a typing environment Γ. Then, we decide if $t \sim_\Gamma u$ as explained bellow:

1. If t, u are not $\to$-normal, we first normalize them. In the following, t and u denote $\to$-normal terms.
2. Then comes the top-down algorithm:
   a) Assume that t and u are in $\mathcal{O}^-$ and $\mathcal{T}, \mathrm{Eq}(\Gamma) \models \bot$. Then t and u are $\sim_\Gamma$-convertible.
   b) Assume that t and u have a non-empty algebraic cap, $\mathcal{T}, \mathrm{Eq}(\Gamma) \not\models \bot$, t has k aliens $a_1, \ldots, a_k$ at positions $p_1, \ldots, p_k$ and u has l aliens $a_{k+1}, \ldots, a_{k+l}$ at positions $p_{k+1}, \ldots, p_{k+n}$. For any alien a_i, we assign a fresh variables c_i s.t. $c_i = c_j$ if and only if a_i is $\sim_\Gamma$ convertible to a_j. Of course, checking $a_i \sim_\Gamma a_j$ is done recursively, by doing all possible pairwise comparisons of the aliens. Then, we obtain two pure algebraic terms $t' = t[p_1 \leftarrow c_1, \ldots, p_k \leftarrow c_k]$ and $u' = u[p_{k+1} \leftarrow c_{k+1}, \ldots, p_{k+l} \leftarrow c_{k+l}]$. t and u are $\sim_\Gamma$-convertible if and only if $\mathcal{T}, \mathrm{Eq}(\Gamma) \models t' = u'$.
   c) In all other cases, we check whether t and u have the same head symbol. If so, we call the procedure recursively on the subterms. Of course, when traversing a binder (with annotation r) binding a pure algebraic equation, it is added to the set of extracted equations. □

5.2 A syntax oriented typing judgment

We now start with the definition of the syntax oriented typing judgment $\vdash_i$.

Definition 5.21 — **Typing judgment $\vdash_i$**

Typing judgment $\vdash_i$ is defined by rules of Figures 5.3 and 5.4.

$$\frac{}{\vdash_i \star : \square}\ [\text{Ax-1}]$$

$$\frac{\Gamma \vdash_i T : s_T \quad \Gamma, [x : T] \vdash_i U : s_U}{\Gamma \vdash_i \forall (x : T).\, U : s_U}\ [\text{Prod}]$$

$$\frac{\Gamma \vdash_i \forall (x : T).\, U : s \quad \Gamma, [x : T] \vdash_i u : U}{\Gamma \vdash_i \lambda [x : T].\, u : \forall (x : T).\, U}\ [\text{Lam}]$$

$$\frac{\Gamma \vdash_i V : s \quad \Gamma \vdash_i t : T \quad s \in \{\star, \square\} \quad x \in \mathcal{X}^s - \mathrm{dom}(\Gamma)}{\Gamma, [x : V] \vdash_i t : T}\ [\text{Weak}]$$

$$\frac{x \in \mathrm{dom}(\Gamma) \cap \mathcal{X}^{s_x} \quad \Gamma \vdash_i x\Gamma : s_x}{\Gamma \vdash_i x : x\Gamma}\ [\text{Var}]$$

$$\frac{\begin{array}{c}\Gamma \vdash_i t : T \quad \Gamma \vdash_i u : U' \quad U \sim_\Gamma U' \\ T_\downarrow = \forall (x :^u U).\, V \\ u \text{ is a weak term} \\ \text{if } x \in \mathcal{X}^-_\star, \text{ then } u \text{ must be in } \mathcal{O}^+ \\ \text{if } x \in \mathcal{X}^-_\square, \text{ then } u \text{ must be in } \mathcal{P}^+\end{array}}{\Gamma \vdash_i t\, u : V\{x \mapsto u\}}\ [\text{App}]$$

Figure 5.3: CCIC Typing Rules for $\vdash_i$ (CC rules)

Proof of correctness and completeness are immediate.

Lemma 5.22 — **Correctness**

If $\Gamma \vdash_i t : T$, then $\Gamma \vdash t : T$

Proof. Direct induction on $\Gamma \vdash_i t : T$, using a one-to-one mapping from rules of $\vdash_i$ to rules of $\vdash$, a extra conversion being needed for the [App] case of $\vdash_i$. □

Lemma 5.23 — **Completeness**

If $\Gamma \vdash t : T$, then there exists T' s.t. i) $\Gamma \vdash_i T' : s$ if $T' \neq \square$, and ii) $T \sim_\Gamma T'$

Proof. If $T = \square$, then we take $T = T' = \square$. Otherwise, we do an induction on $\Gamma \vdash t : T$. The only delicate case are the [App] and [Conv] ones.

- If $\Gamma \vdash t : T$ is deduced from [App], then $t = u\,v : V\{x \mapsto u\}$ with i) $\Gamma \vdash t : \forall (x :^u U).\, V$, ii) if $x \in \mathcal{X}^-_\star$, then $v \in \mathcal{O}^+$, iii) v is weak, and iv) $\Gamma \vdash u : U$.

  By induction hypothesis, there exists P and U' s.t. $\Gamma \vdash_i v : P$ and $\Gamma \vdash_i u : U'$, with $P \sim_\Gamma \forall (x :^a U).\, V$, $U' \sim_\Gamma U$.

$$A = \forall(\overrightarrow{x : T}).\star \quad \vdash_i A : \square \quad \text{for all } i,\ \Gamma \vdash_i C_i : \star$$
$$\text{for all } i,\ C_i \text{ is a strictly positive constructor in } X$$
$$\frac{I = \mathrm{Ind}(X : A)\{\overrightarrow{C_i}\} \text{ is in } \xrightarrow{\beta\iota}\text{-normal form}}{\Gamma \vdash_i I : T}\ [\textsc{Ind}]$$

$$\frac{I = \mathrm{Ind}(X : T)\{\overrightarrow{C_i}\} \quad \Gamma \vdash_i I : T}{\Gamma \vdash_i I^{[k]} : C_k\{X \mapsto I\}}\ [\textsc{Constr}]$$

$$A = \forall(\overrightarrow{x : U}).\star \quad I = \mathrm{Ind}(X : T)\{\overrightarrow{C_j}\} \quad \Gamma \vdash_i I : T \quad \vdash_i Q : \forall(\overrightarrow{x : U}).\,(I\ \vec{x}) \to \star$$
$$T_i = \Delta^{\star}\{I, X, C_i, Q, I^{[i]}\} \quad \vdash_i T_i : \star$$
$$\frac{\text{for all } j,\ \Gamma \vdash_i a_j : A_j\{\overrightarrow{x \mapsto a}\} \quad \Gamma \vdash_i c : I\ \vec{a} \quad \text{for all } i,\ \Gamma \vdash_i f_i : T_i}{\Gamma \vdash_i \mathrm{Elim}(c : I\,[\vec{a}] \to Q)\{\vec{f}\} : Q\ \vec{a}\ c}\ [\textsc{Elim-}\star]$$

$$A = \forall(\overrightarrow{x : U}).\star \quad I = \mathrm{Ind}(X : T)\{\overrightarrow{C_j}\} \text{ is small}$$
$$Q = \forall(\overrightarrow{x : U})(y : I\ \vec{x}).\,K \text{ is in } \xrightarrow{\beta\iota}\text{-normal form}$$
$$[\overrightarrow{x : U}], [y : I\ \vec{x}] \vdash_i K : \square$$
$$T_i = \Delta^{\square}\{I, X, C_i, \vec{x}\ y, K, I^{[i]}\} \quad \vdash_i T_i : \square$$
$$\frac{\text{for all } j,\ \Gamma \vdash_i a_j : A_j\{\overrightarrow{x \mapsto a}\} \quad \Gamma \vdash_i c : I\ \vec{a} \quad \text{for all } i,\ \Gamma \vdash_i f_i : T_i}{\Gamma \vdash_i \mathrm{Elim}(c : I\,[\vec{a}] \to Q)\{\vec{f}\} : K\{\overrightarrow{x \mapsto}, y \mapsto c\}}\ [\textsc{Elim-}\square]$$

Figure 5.4: CCIC Typing Rules for $\vdash_i$ (Inductive Types)

By type structure compatibility, $P_\downarrow = \forall(x :^a U_\downarrow).V_\downarrow$. Hence, $U' \sim_\Gamma U \sim_\Gamma U_\downarrow$. We clearly have U and thus $U_\downarrow$ well-formed. From correctness of $\vdash_i$, we have $\Gamma \vdash u : U'$ and hence, U' is also well-formed. (U' cannot be the sort $\square$ since $\Gamma \vdash u : U$ and U is well-formed) Hence, by transitivity of $\sim_\Gamma$ on well-formed terms, $U' \sim_\Gamma U_\downarrow$.

We can then apply the [App] rule for $\vdash_i$, obtaining that $\Gamma \vdash_i V'\{x \mapsto u\}$, with $V'\{x \mapsto u\} \sim_\Gamma V\{x \mapsto u\}$ from substitutivity lemma.

- If $\Gamma \vdash t : T$ from [Conv], then $\Gamma \vdash t : U$ with $U \sim_\Gamma T$, and by application of the induction hypothesis, there exists a term V s.t. $\Gamma \vdash_i t : V$ with $V \sim_\Gamma U \sim_\Gamma T$. Since $U \neq \square$ (U is well-formed from rule assumption), $U \neq \square$. From correctness of $\vdash_i$, we have $\Gamma \vdash t : V$ and $V \neq \square$ since $V \sim_\Gamma U$. Hence, V is well-formed too and we obtain $V \sim_\Gamma T$ from transitivity of $\sim_\Gamma$ on well-formed terms. □

5.3 Deciding more theories

We now explain the extension of our algorithm for arbitrary theories, using as example the theory composed of the parametric lists and the Presburger arithmetic. The main difficulty resides in the [Ded] rule, which algebraised all the convertible terms for all the possible first-order sorts.

We start with the definition of $\mathcal{O}^+$. Taking here the set of pure algebraic terms (defined as the set of terms having no aliens w.r.t. a first-order sort):

$$\mathcal{O}^+ = \{t \mid \exists \sigma \in \Lambda_{\mathcal{E}}.\ \mathrm{FV}(\mathcal{A}_{\mathcal{R}}(t)(\sigma)) \cap \mathcal{Y} = \varnothing\}$$

is of no use since it is not stable by substitution. For example, $x \dot{+} \mathbf{0}$ and $\underline{\mathbf{nil}}\, A$ are in $\mathcal{O}^+$, but not $(\underline{\mathbf{nil}}\, A) \dot{+} \mathbf{0}$. What breaks here the stability by substitution is the possibility of constructing ill-formed terms by well-typed substitution. Taking this into account yields a definition of $\mathcal{O}^+$ where instead of considering pure algebraic terms, we consider terms having a non-empty algebraic cap and composed only of subterms having non-algebraic caps. I.e. terms of the form:

$$t, u, \ldots ::= x \in \mathcal{X}^-_\star \mid \mathbf{0} \mid \mathbf{S}\, t \mid t \dot{+} u \mid \underline{\mathbf{nil}}\, A \mid \underline{\mathbf{cons}}\, A\, e\, l \mid \underline{\mathbf{app}}\, A\, l_1\, l_2.$$

Note that restricting this definition to the case of Presburger arithmetic yield our previous notion of $\mathcal{O}^+$.

The set $\mathcal{O}^+$ is now clearly stable by well-sorted substitutions and stable by reduction. For the stability of $\mathcal{O}^+$ w.r.t. conversion (property 7 of Definition 4.1) to hold, we must add the two following rewrite rules:

$$\underline{\mathbf{car}}\, A\, (\underline{\mathbf{cons}}\, B\, x\, l) \xrightarrow{R} x$$
$$\underline{\mathbf{cdr}}\, A\, (\underline{\mathbf{cons}}\, B\, x\, l) \xrightarrow{R} l$$

so that when having a conversion of the form $x \sim_\Gamma \underline{\mathbf{car}}\, A\, (\underline{\mathbf{cons}}\, B\, x\, l)$ with l not in $\mathcal{O}^+$, then $\underline{\mathbf{car}}\, A\, (\underline{\mathbf{cons}}\, B\, x\, l) \to x$.

For $\mathcal{P}^+$, we do not change our definition. Extended to the case of Presburger arithmetic and parametric lists, we obtain the set $\mathcal{P}^+$ composed of terms convertible to terms of the form:

$$T, U, \ldots ::= A \in \mathcal{X}^-_\square \mid \underline{\mathbf{nat}} \mid \underline{\mathbf{list}}\, T$$

We now come to the decidability of the [Ded] and [Eq] rules. As for the Presburger arithmetic, [Eq] is replaced by an a priori extraction of equations: this is the set $\mathrm{Eq}(\Gamma)$. We do not change its definition:

$$\mathrm{Eq}(\Gamma) = \{t_1 \doteq t_2 \mid [x :^r T] \in \Gamma, T \to_* t_1 \doteq t_2, t_1, t_2 \in \mathcal{O}^+\}$$

Note that now, $\mathrm{Eq}(\Gamma)$ is not composed of pure algebraic equations. Assume now that Γ is a typing environment containing the two extractable equations:

$$\mathbf{cons}\, A\, x_A\, l_A \doteq \mathbf{cons}\, B_1\, x_B\, l_B \qquad (E_1)$$
$$\mathbf{cons}\, B_2\, x_B\, l_B \doteq \mathbf{cons}\, C\, x_C\, l_C \qquad (E_2)$$

Hence, $x_A \sim_\Gamma x_C$. Indeed, assuming that $\sigma = \mathbf{list}(\alpha)$, then the algebraisation of (E_1) and (E_2) w.r.t. σ yields the two equations

$$\mathbf{cons}(y_A, z_A) = \mathbf{cons}(y_B, y_B) \text{ and } \mathbf{cons}(y_B, z_B) = \mathbf{cons}(y_C, z_C)$$

with $y_A = \mathcal{A}_{\sim_\Gamma}(x_A)(\alpha)$ (resp. $y_B = \mathcal{A}_{\sim_\Gamma}(x_B)(\alpha)$, $y_C = \mathcal{A}_{\sim_\Gamma}(x_C)(\alpha)$). Hence $y_A = y_C$ is valid in the theory and $x_A \sim_\Gamma x_C$ by [Ded]. Note that the algebraisations of (E_1) and (E_2) w.r.t. other first-order sorts if of no use: they will either give the same first-order equations (up to a renaming), or worst, will simply yield weaker equations (e.g. when algebraizing (E_1) and (E_2) w.r.t. the sort **nat**).

Hence, an equation of the form $l \doteq \mathbf{cons}\,\underline{\mathbf{nat}}\,(\mathbf{nil}\,A)\,l'$ will only be algebraised using the two sorts $\mathbf{list}(\alpha)$ and **nat**. The first one because the first element of the list is headed by $\underline{\mathbf{nil}}\,A$, the second one because the type parameter of the list if $\underline{\mathbf{nat}}$.

Finally, $t \approx_\Gamma u$ for t or u having a non-algebraic cap is extended as follow. Let E be the set of all the possible algebraisation of $\mathrm{Eq}(\Gamma)$ (which is now finite), and let $t' = u'$ be a possible algebraisation of the equation $t \doteq u$. Note that when choosing a possible algebraisation of $t \doteq u$, we make here a non-deterministic choice. As for the Presburger case, the algebraisation is now done by a pairwise comparison, w.r.t. $\approx_\Gamma$, of all the encountered aliens (i.e. the aliens of t and u as well as the aliens of all the terms of E). Then, we conclude $t \approx_\Gamma u$ if $\mathcal{T}, E \models t' = u'$ holds.

However, it is easy to see that there are no non-deterministic choices when checking conversion w.r.t. a well-formed environment and well-formed terms. Indeed, a non-deterministic choice occurs when several arguments of a function symbol disagree. This is the case of the ill-formed term $\mathbf{cons}\,\underline{\mathbf{nat}}\,(\mathbf{nil}\,A)\,l$ where the type parameter indicates a list of type $\underline{\mathbf{nat}}$ but the first parameter is of type $\underline{\mathbf{list}}\,A$. Now, having well-formed terms should lead to unique algebraisation. For example, $\mathbf{cons}\,\underline{\mathbf{nat}}\,\mathbf{0}\,l$ will be algebraised as a list of type $\underline{\mathbf{nat}}$ where $\mathbf{cons}\,(\underline{\mathbf{list}}\,A)\,(\underline{\mathbf{nil}}\,B)\,l$ will be algebraised as a list of list, as long as A is convertible to B, which is the case when the term is well-formed.

CHAPTER 6

Conclusion and perspectives

In this thesis, we have described how decision procedures for first-order theories over equality can be introduced into the conversion relation of the Calculus of Inductive Constructions. We have shown that this extension does not break logical consistency of the calculus as well as the decidability of type-checking.

To this end, we have introduced strong syntactical restrictions which limit the benefit of such an extension. We review here them all and give suggestions for their removal.

We also point out several research directions.

6.1 Stability of extractable equations

Equations in CCIC are introduced via lambda abstractions and dependent products. This is the case in the term $\lambda[p :^r x \doteq y].\, t$ where the typing of the subterm t may make use of the equation $x \doteq y$. As have seen however, this way of extracting equations does not behave well with $\xrightarrow{\beta}$-reduction, because some equality predicates (the type $x \doteq y$ of p in our example) are erased.

To this end, two annotations were introduced indicating which equations can be used by conversion and which cannot: in $\lambda[p :^a x \doteq y].\, t$, the equation $x \doteq y$ can be used only if $p :^a x \doteq y$ is annotated with the restricted annotation, that is $a = r$. Moreover, application of r-annotated λ-abstractions are forbidden, therefore eliminating the problem of erasing equality predicates.

The idea is then to make use of definitions, as available in the Coq system. A *definition* is a triple $x : T := t$ where x is a variable, T the type associated to x and t a term of type T. Definitions are introduced using a new typing rule

$$\frac{\Gamma \vdash t : T}{\Gamma, [x : T := t] \vdash x : T}$$

There are two kinds of definitions in Coq, called respectively *global* and *opaque*.

To a global definition $x : T := t$ is associated a reduction relation $\xrightarrow{\delta,\Gamma}$ such that $x \xrightarrow{\delta,\Gamma} t$ if $x : T := t$ appears in Γ - we say that x has been unfolded. It is known that adding global definitions is harmless (the case of local definitions is harder and studied in [40]).

On the other hand, *opaque* definitions cannot be unfolded, that is, there are no rules associated to them. They conform to the mathematical tradition of proof irrelevance, in which proofs of a logical propositions do not matter. Hence, although the term $(\lambda[p :^r x \doteq y].\, t)\, q$ is not well-sorted in the present version of CCIC, it could be expressed as the application $u\, q$ where u is an opaque definition for $\lambda[p :^r x \doteq y].\, t$, of type $\forall(p :^r x \doteq y).\, T$.

An alternative to opaque definition to overcome this problem would be the use of term annotations keeping track of all equations used in a conversion: terms would be annotated

with a sequence of pairs (e, t) where e is an extractable equation and t a term of type e in the current environment. $\xrightarrow{\beta}$-reduction would then be modified as follows:

$$(\lambda[x :^{r} T].u)\, v \xrightarrow{\beta} u\{x \mapsto v\} \oplus (T, v)$$

where $p \oplus (T, v)$ is the term p whose annotation is augmented with the pair (T, v).

Conversion could then be modified in order to use the equations present in annotated terms, hence ensuring the stability of extractable equation w.r.t. $\xrightarrow{\beta}$-reduction. In this modified calculus, the restricted annotation would no more be used to restrict the application of terms, but as a hint to the system to select which extractable equations it should use in a given conversion goal.

6.2 Using a typed extraction

In order to ensure stability of extractable equations by substitution, and to make type--checking decidable, strong syntactical restrictions have been made: a set $\mathcal{X}^{-}_{\star}$ of extractable variables has been introduced and only those substitutions mapping extractable variables to extractable terms are allowed.

Instead, we could have defined a notion of extractable terms w.r.t. the types of the terms - hence, obtaining a typed conversion. For example, one could decide to extract from an environment Γ those equations only which related terms of type **nat** in the considered environment.

In such a calculus, stability of extractable equations w.r.t. well-formed substitutions becomes a special case of substitutivity: the set of extractable variables as well as the syntactical restrictions on substitutions become useless. Further, we think that this calculus enjoys a decidable type-checking problem as well. To substantiate this belief, let us remark that in the current version of CCIC, $\mathcal{O}^{+}$ is restricted to be the set of pure algebraic terms so that the equation

$$\lambda[x : \mathbf{nat}].\, f\, x \doteq \lambda[x : \mathbf{nat}].\, f\, (x \dot{+} \mathbf{2})$$

cannot be extracted. Indeed, we saw that allowing such an equation introduces an encoding of universally quantified first-order formulas into the deductional part of our conversion, resulting in the undecidability of type-checking. Restricting extractable equations to the ones relating terms of type **nat** also forbids the use of such equations as long as product compatibility holds.

6.3 Weak terms

Weak terms were introduced in order to limit the interactions between the deductive part of conversion and the recursors. We saw that allowing conversion under strong recursors may yield the convertibility of A and $A \to A$ under an inconsistent environment. On the other hand, allowing conversion under weak recursors is important for practice, allowing a kind of non-structural elimination. For example, one could expect the following reduction to hold, as long as x is convertible to $\mathbf{0}$:

$$\mathrm{Elim}(x : \mathbf{nat}\,[\epsilon] \to Q)\{[f_0, f_S]\} \xrightarrow{\iota} f_0.$$

Because of the strong dependency between extractable and convertible terms - as exemplified by the requirement that extractable terms must be stable by conversion -, we know that allowing conversions below weak recursors will require extracting equations containing weak recursors that cannot be eliminated by $\xrightarrow{\iota}$-reduction alone. Indeed, any term of the form $\mathrm{Elim}(x : I\,[\vec{u}] \to Q)\{\vec{f}\}$, with f_0 algebraic, must become extractable, since

$$\mathrm{Elim}(x : I\,[\vec{u}] \to Q)\{\vec{f}\} \sim_\Gamma \mathrm{Elim}(\mathbf{0} : I\,[\vec{u}] \to Q)\{\vec{f}\} \xrightarrow{\iota} f_0$$

in any typing environment s.t. $x \sim_\Gamma \mathbf{0}$.

A main reason why type-checking of CCIC is decidable is the strict separation between $\to$-conversion and the deductive part of the embedded first-order theory. We could however imagine an approach where variables could appear under weak recursors of extractable equations as long as they do not appear in subsequent equations. This would allow extracting the equation

$$y = \mathrm{Elim}(x : I\,[\vec{u}] \to Q)\{\vec{f}\} \qquad (*)$$

as long as no extractable equation involving x appears in the rest of the environment.

In this case, there is no strict separation anymore between conversion and $\to$-conversion, since the presence of the extractable equation $x = \mathbf{0}$ occurring before $(*)$ in the typing environment would lead to the conversion $y \sim_\Gamma f_0$. We hope nevertheless to find an incremental decision procedure, using the underlying order over variables induced by the suggested restriction.

6.4 Extending CCIC to CAC

One important motivation of CAC was the introduction of *type level* rewriting in the conversion rule, which permits, for example, the introduction of decision procedures for first order tautologies.

One can think of a calculus merging the embedding of first-order theories over equations as well as type level rewriting (besides the already existing strong $\xrightarrow{\iota}$-reduction).

We believe that this could be done by extending the notion of weak terms so as to take into account the new interactions between type level rewriting and the embedded theory.

6.5 Embedding a more powerful logic

Our choice of embeddable theories have some drawbacks. For example, it is not possible to use the first-order theory of lists for the CCIC type of dependent lists, the mapping between first-order signatures and CCIC symbols being strict, and thus not allowing the use of extra parameters (here the length of the list) in the type of mapped function symbols.

The user is then facing a choice: to use the non-dependent type of lists, and benefit from the first-order theory of lists in conversions, or to use the dependent type of lists without having the first-order theory of lists available in conversions, but having instead the theory of linear arithmetic available for converting the dependent arguments.

Several directions can be investigated to overcome this problem and have the benefit of both in the case of dependent lists. A first is to allow embedding the more expressive (parametric version of) membership equational logic [7] in CCIC along the lines of the simpler embedding described here. A second is to consider the case of dependent algebras instead of the simpler parametric algebras. This is a more difficult question, which requires using our generalized notion of conversion in the main argument of an elimination, but would further help us addressing other weaknesses of Coq.

Bibliography

[1] B. Barras. *Auto-Validation d'un système de preuves avec familles inductives.* PhD thesis, Université de Paris VII, 1999.

[2] G. Barthe. The relevance of proof-irrelevance. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 755–768. Springer, 1998.

[3] F. Blanqui. *Type Theory and Rewriting.* PhD thesis, Universite Paris XI, 2001.

[4] F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.

[5] F. Blanqui. Inductive types in the calculus of algebraic constructions. *Fundam. Inform.*, 65(1-2):61–86, 2005.

[6] F. Blanqui, J.-P. Jouannaud, and P.-Y. Strub. Building decision procedures in the calculus of inductive constructions. In J. Duparc and T. A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 328–342. Springer, 2007.

[7] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theor. Comput. Sci.*, 236(1-2):35–132, 2000.

[8] A. Church. A simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[9] R. Constable, S. Allen, M. Bromley, R. Cleaveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. Sasaki, and S. Smith. *Implementing mathematics with the Nuprl proof development system.* Prentice Hall, 1986.

[10] Coq Development Team. The Coq proof assistant - version 8. Technical report, INRIA, 2008.

[11] T. Coquand. An analysis of girard's paradox, 1986.

[12] T. Coquand. *An algorithm for testing conversion in type theory.* Cambridge University Press, New York, NY, USA, 1991.

[13] T. Coquand and G. Huet. The calculus of constructions. *Inf. Comput.*, 76(2-3):95–120, 1988.

[14] T. Coquand and G. P. Huet. The calculus of constructions. *Inf. Comput.*, 76(2/3):95–120, 1988.

[15] T. Coquand and C. Paulin. Inductively defined types. In *COLOG-88: Proceedings of the international conference on Computer logic*, pages 50–66, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[16] H. Curry and R. Feys. *Combinatory Logic I.* Studies in Logic and the Foundations of Mathematics. North-Holland, 1958. Volume II, with Jonathan Seldin, 1972.

[17] N. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger, editors, *Proc. of Symp. on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61. Springer-Verlag, 1970.

[18] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *J. Autom. Reasoning*, 31(1):33–72, 2003.

[19] G. Dowek and B. Werner. Proof normalization modulo. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *TYPES*, volume 1657 of *Lecture Notes in Computer Science*, pages 62–77. Springer, 1998.

[20] G. Gentzen. Untersuchungen über das logisches schließen. *Mathematische Zeitschrift*, 1:176–210, 1935.

[21] H. Geuvers and M.-J. Nederhof. Modular proof of strong normalization for the calculus of constructions. *J. Funct. Program.*, 1(2):155–189, 1991.

[22] E. Gimenez. Structural recursive definitions in type theory. In *Automata, Languages and Programming*, pages 397–408, 1998.

[23] J.-Y. Girard. Une extension de l'interprétation de gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. Fenstad, editor, *Proc. of the 2nd Scandinavian Logic Symposium*, volume 63. North-Holland, 1971.

[24] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1988.

[25] G. Gonthier. The four color theorem in coq. In *TYPES 2004 International Workshop*, 2004.

[26] M. Gordon and T. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge Univ. Press, 1993.

[27] K. Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme. *Monatshefte für Mathematik und Physik*, 38:163–198, 1931.

[28] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1969.

[29] J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *LICS*, pages 350–361. IEEE Computer Society, 1991.

[30] Z. Luo. ECC, an extended calculus of constructions. In *Proceedings 4th Annual IEEE Symp. on Logic in Computer Science, LICS'89, Pacific Grove, CA, USA, 5–8 June 1989*, pages 386–395. IEEE Computer Society Press, Los Alamitos, CA, 1989.

[31] P. Martin-Löf. Haupsatz for the intuitionistic theory of iterated inductive definitions. In J. Fenstad, editor, *Proc. of the 2nd Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1971. See [32].

[32] P. Martin-Löf. *Intuitionistic type theory*. Bibliopolis, 1984.

[33] R. Milner. Implementation and applications of Scott's logic for computable functions. *ACM sigplan notices*, 7(1):1–6, Jan. 1972.

[34] M. Okada. Strong normalizibility for the combined system of the typed lambda calculus and arbitrary convergent term rewriting system. In *Proc. of the 1989 Int. Symp. on Symbolic and Algebraic Computation*. ACM Press, 1989.

[35] N. Oury. Extensionality in the calculus of constructions. In J. Hurd and T. F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2005.

[36] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Proc. 11th Int. Conf. on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, June 1992.

[37] L. Paulson. Experience with Isabelle : A generic theorem prover. Technical Report UCAM-CL-TR-143, University of Cambridge, Computer Laboratory, Aug. 1988.

[38] L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[39] D. Prawitz. *Natural Deduction: a Proof-Theoretical Study*, volume 3 of *Stockholm Studies in Philosophy*. Almqvist & Wiksell, 1965.

[40] P. Severi and E. Poll. Pure type systems with definitions. In A. Nerode and Y. Matiyasevich, editors, *LFCS*, volume 813 of *Lecture Notes in Computer Science*, pages 316–328. Springer, 1994.

[41] N. Shankar. Little engines of proof. In L.-H. Eriksson and P. Lindsay, editors, *FME 2002: Formal Methods — Getting IT Right, Copenhagen*, pages 1–20. Springer-Verlag, 2002.

[42] N. Shankar, S. Owre, J. Rushby, and D. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Sept. 1999.

[43] R. E. Shostak. Deciding combinations of theories. *J. ACM*, 31(1):1–12, 1984.

[44] V. Tannen. Combining algebra and higher-order types. In *LICS*, pages 82–90. IEEE Computer Society, 1988.

[45] V. Tannen and J. H. Gallier. Polymorphic rewriting conserves algebraic strong normalization and confluence. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 137–150. Springer, 1989.

[46] V. Tannen and J. H. Gallier. Polymorphic rewriting conserves algebraic confluence. *Inf. Comput.*, 114(1):1–29, 1994.

[47] A. Trybulec. The Mizar-QC/6000 logic information language. In *Association for Literary and Linguistic Computing Bulletin*, volume 6, pages 136–140, 1978.

[48] B. Werner. *Une théorie des Constructions Inductives*. PhD thesis, Université de Paris VII, 1994.